Nonlinear differential equation problems

Hans Petter Langtangen

edical Computing, Simula Research Laboratory and Department of Informatics, Universi

Nov 5, 2014

Note: This note is still under development.

Contents

In a linear differential equation all terms involving the unknown fu
are linear in the unknown functions or their derivatives. Linear here mea
the unknown function or a derivative of it is multiplied by a number or a
function. All other differential equations are non-linear. The easiest way
an equation is nonlinear is to spot nonlinear terms where the unknown fu
or their derivatives are multiplied by each other. For example, in

$$u'(t) = -a(t)u(t) + b(t),$$

the terms involving the unknown function u are linear: u' contains the de
of the unknown function multiplied by unity, and au contains the u
function multiplied by a known function. However,

$$u'(t) = u(t)(1 - u(t)),$$

is nonlinear because of the term $-u^2$ where the unknown function is mu
by itself. Also

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0,$$

is nonlinear because of the term uu_x where the unknown function app
a product with itself or one if its derivatives. Another example of a no
equation is

$$u'' + \sin(u) = 0,$$

because $\sin(u)$ contains products of u,

$$\sin(u) = u - \frac{1}{3}u^3 + \ldots$$

A series of forthcoming examples will explain who to tackle nonlinea
ential equations with various techniques.

Introduction of basic concepts

onsider the (scaled) logistic equation

$$u'(t) = u(t)(1 - u(t))\,. \tag{1}$$

his is a nonlinear differential equation which will be solved by different strategies ı the following. A time discretization of (1) will either lead to a linear algebraic quation or a nonlinear algebraic equation at each time level. In the former ase, the time discretization method transforms the nonlinear ODE into linear ıbproblems at each time level, and the solution is straightforward to find nce linear algebraic equations are easy to solve by hand. However, when the me discretization leads to nonlinear algebraic equations, we cannot (except in ery rare cases) solve these without turning to approximate, iterative solution ıethods.

The following subsections first introduce various methods using (1):

- explicit time discretization methods (with no need to solve nonlinear algebraic equations)
- implicit Backward Euler discretization, leading to nonlinear algebraic equations solved by
  - an exact analytical technique
  - Picard iteration based on manual linearization
  - a single Picard step
  - Newton's method
- Implicit Crank-Nicolson discretization and linearization via a geometric mean formula

hereafter, we compare the performance of the various approaches. Despite the mplicity of (1), the conclusions reveal typical features of the various methods ı much more complicated nonlinear PDE problems.

.1 Linearization by explicit time discretization

Forward Euler method to solve (1) results in

$$\frac{u^{n+1} - u^n}{\Delta t} = u^n(1 - u^n),$$

hich is a *linear* algebraic equation for the unknown value u^{n+1}. The nonlinearity ı the original equation poses in this case no difficulty in the discrete algebraic quation. Any other explicit scheme in time will also give only linear algebraic quations to solve. For example, a typical 2nd-order Runge-Kutta method for .) reads,

$$u^* = u^n + \Delta t u^n(1 - u^n),$$
$$u^{n+1} = u^n + \Delta t \frac{1}{2}\left(u^n(1 - u^n) + u^*(1 - u^*)\right)\,.$$

The first step is linear in the unknown u^*. Then u^* is known in the ne which is linear in the unknown u^{n+1} .

1.2 Exact solution of nonlinear equations

Switching to a Backward Euler scheme for (1),

$$\frac{u^n - u^{n-1}}{\Delta t} = u^n(1 - u^n),$$

results in a nonlinear algebraic equation for the unknown value u^n. The e is of quadratic type:

$$\Delta t(u^n)^2 + (1 - \Delta t)u^n - u^{n-1} = 0\,.$$

We shall now introduce a shorter and often cleaner notation for nc algebraic equations at a given time level. The notation is inspired by the notation, i.e., variable names, used in a program, especially in more ac partial differential equation problems. The unknown in the algebraic equ denoted by u, while $u^{(1)}$ is the value of the unknown at the previous tir (in general $u^{(\ell)}$ is the value of the unknown ℓ levels back in time). The ı will be frequently used in later sections. What is meant by u (the exact of the PDE problem, the numerical approximation to the exact solution unknown solution at a certain time level) should be evident from the cc

The quadratic equation for the unknown u^n in (2) can with the new ı be written

$$F(u) = \Delta t u^2 + (1 - \Delta t)u - u^{(1)} = 0\,.$$

The solution is readily found to be

$$u = \frac{1}{2\Delta t}\left(-1 - \Delta t \pm \sqrt{(1 - \Delta t)^2 - 4\Delta t u^{(1)}}\right)\,.$$

Now we encounter a fundamental challenge with nonlinear algebraic eq the equation may have more than one solution. How do we pick the right s In the present simple case we can expand the square root in a series in truncate after the linear term since the Backward Euler scheme will in an error proportional to Δt anyway. Using `sympy` we find the following series expansions of the roots:

```
>>> import sympy as sp
>>> dt, u_1, u = sp.symbols('dt u_1 u')
>>> r1, r2 = sp.solve(dt*u**2 + (1-dt)*u - u_1, u)  # find roots
>>> r1
(dt - sqrt(dt**2 + 4*dt*u_1 - 2*dt + 1) - 1)/(2*dt)
>>> r2
(dt + sqrt(dt**2 + 4*dt*u_1 - 2*dt + 1) - 1)/(2*dt)
>>> print r1.series(dt, 0, 2)
-1/dt + 1 - u_1 + dt*(u_1**2 - u_1) + O(dt**2)
>>> print r2.series(dt, 0, 2)
u_1 + dt*(-u_1**2 + u_1) + O(dt**2)
```

We see that the `r1` root, corresponding to a minus sign in front of the square root in (4), behaves as $1/\Delta t$ and will therefore blow up as $\Delta t \to 0$! Therefore, only the `r2` root is of relevance in this case.

.3 Linearization

When the time integration of an ODE results in a nonlinear algebraic equation, we must normally find its solution by defining a sequence of linear equations and hope that the solutions of these linear equations converge to the desired solution of the nonlinear algebraic equation. Usually this means solving the linear equation repeatedly in an iterative fashion. Alternatively, the nonlinear equation can sometimes be approximated by one linear equation, and consequently there is no need for iteration.

Constructing a linear equation from a nonlinear one requires *linearization* of each nonlinear term. This can be done manually as in Picard iteration, or fully algorithmically as in Newton's method. Examples will best illustrate how to linearize nonlinear problems.

.4 Picard iteration

Let us write (3) in a more compact form

$$F(u) = au^2 + bu + c = 0,$$

with $a = \Delta t$, $b = 1 - \Delta t$, and $c = -u^{(1)}$. Let u^- be an available approximation of the unknown u. Then we can linearize the term u^2 simply by writing $u^- u$. The resulting equation, $\hat{F}(u) = 0$, is now linear and hence easy to solve:

$$F(u) \approx \hat{F}(u) = au^- u + bu + c = 0\,.$$

Since the equation $\hat{F} = 0$ is only approximate, the solution u does not equal the exact solution u_e of the exact equation $F(u_\mathrm{e}) = 0$, but we can hope that u is closer to u_e than u^- is, and hence it makes sense to repeat the procedure, i.e., set $u^- = u$ and solve $\hat{F}(u) = 0$ again.

The idea of turning a nonlinear equation into a linear one by using an approximation u^- of u in nonlinear terms is a widely used approach that goes under many names: *fixed-point iteration*, the method of *successive substitutions*,

nonlinear Richardson iteration, and *Picard iteration*. We will stick to th name.

Picard iteration for solving the nonlinear equation arising from the Ba Euler discretization of the logistic equation can be written as

$$u = -\frac{c}{au^- + b}, \quad u^- \leftarrow u\,.$$

The iteration is started with the value of the unknown at the previous tim $u^- = u^{(1)}$.

Some prefer an explicit iteration counter as superscript in the mathe notation. Let u^k be the computed approximation to the solution in iter In iteration $k+1$ we want to solve

$$au^k u^{k+1} + bu^{k+1} + c = 0 \quad \Rightarrow \quad u^{k+1} = -\frac{c}{au^k + b}, \quad k = 0, 1, \ldots$$

However, we will normally apply a mathematical notation in our final f that is as close as possible to what we aim to write in a computer code a it becomes natural to use u and u^- instead of u^{k+1} and u^k.

Stopping criteria. The iteration method can typically be terminate the change in the solution is smaller than a tolerance ϵ_u:

$$|u - u^-| \leq \epsilon_u,$$

or when the residual in the equation is sufficiently small (ϵ_r),

$$|F(u)| = |au^2 + bu + c| < \epsilon_r\,.$$

A single Picard iteration. Instead of iterating until a stopping crit fulfilled, one may iterate a specific number of times. Just one Picard iter popular as this corresponds to the intuitive idea of approximating a n term like $(u^n)^2$ by $u^{n-1}u^n$. This follows from the linearization $u^- u^n$ initial choice of $u^- = u^{n-1}$ at time level t_n. In other words, a single iteration corresponds to using the solution at the previous time level to l nonlinear terms. The resulting discretization becomes

$$\frac{u^n - u^{n-1}}{\Delta t} = u^n(1 - u^{n-1}),$$

which is a linear algebraic equation in the unknown u^n, and therefore easily solve for u^n, and there is no need for any alternative notation.

We shall later refer to the strategy of taking one Picard step, or equi linearizing terms with use of the solution at the previous time step, as the method. It is a widely used approach in science and technology, but wi limitations if Δt is not sufficiently small (as will be illustrated later).

Notice.

Equation (5) does not correspond to a "pure" finite difference method where the equation is sampled at a point and derivatives replaced by differences (because the u^{n-1} term on the right-hand side must then be u^n). The best interpretation of the scheme (5) is a Backward Euler difference combined with a single (perhaps insufficient) Picard iteration at each time level, with the value at the previous time level as start for the Picard iteration.

.5 Linearization by a geometric mean

/e consider now a Crank-Nicolson discretization of (1). This means that the me derivative is approximated by a centered difference,

$$[D_t u = u(1-u)]^{n+\frac{1}{2}},$$

ritten out as

$$\frac{u^{n+1}-u^n}{\Delta t} = u^{n+\frac{1}{2}} - (u^{n+\frac{1}{2}})^2\,. \tag{6}$$

he term $u^{n+\frac{1}{2}}$ is normally approximated by an arithmetic mean,

$$u^{n+\frac{1}{2}} \approx \frac{1}{2}(u^n + u^{n+1}),$$

ıch that the scheme involves the unknown function only at the time levels where e actually compute it. The same arithmetic mean applied to the nonlinear ɛrm gives

$$(u^{n+\frac{1}{2}})^2 \approx \frac{1}{4}(u^n + u^{n+1})^2,$$

hich is nonlinear in the unknown u^{n+1}. However, using a *geometric mean* for $\iota^{n+\frac{1}{2}})^2$ is a way of linearizing the nonlinear term in (6):

$$(u^{n+\frac{1}{2}})^2 \approx u^n u^{n+1}\,.$$

sing an arithmetic mean on the linear $u^{n+\frac{1}{2}}$ term in (6) and a geometric mean ɔr the second term, results in a linearized equation for the unknown u^{n+1}:

$$\frac{u^{n+1}-u^n}{\Delta t} = \frac{1}{2}(u^n + u^{n+1}) + u^n u^{n+1},$$

hich can readily be solved:

$$u^{n+1} = \frac{1+\frac{1}{2}\Delta t}{1+\Delta t u^n - \frac{1}{2}\Delta t}u^n\,.$$

This scheme can be coded directly, and since there is no nonlinear a equation to iterate over, we skip the simplified notation with u for u^{n+1} for u^n.

The geometric mean approximation is often very effective for lin quadratic nonlinearities. Both the arithmetic and geometric mean app tions have truncation errors of order Δt^2 and are therefore compatible truncation error $\mathcal{O}(\Delta t^)$ of the centered difference approximation for u Crank-Nicolson method.

Applying the operator notation for the means and finite differen linearized Crank-Nicolson scheme for the logistic equation can be co expressed as

$$[D_t u = \overline{u}^t + \overline{u^2}^{t,g}]^{n+\frac{1}{2}}\,.$$

Remark.

If we use an arithmetic instead of a geometric mean for the nonl term in (6), we end up with a nonlinear term $(u^{n+1})^2$. This term c linearized as $u^- u^{n+1}$ in a Picard iteration approach and in particul $u^n u^{n+1}$ in a Picard1 iteration approach. The latter gives a scheme a identical to the one arising from a geometric mean (the difference in being $\frac{1}{4}\Delta t u^n(u^{n+1}-u^n) \approx \frac{1}{4}\Delta t^2 u'u$, i.e., a difference of $\mathcal{O}(\Delta t^2)$).

1.6 Newton's method

The Backward Euler scheme (2) for the logistic equation leads to a n algebraic equation (3). Now we write any nonlinear algebraic equatio general and compact form

$$F(u) = 0\,.$$

Newton's method linearizes this equation by approximating $F(u)$ by its series expansion around a computed value u^- and keeping only the lin

$$\begin{aligned} F(u) &= F(u^-) + F'(u^-)(u-u^-) + \frac{1}{2}F''(u^-)(u-u^-)^2 + \cdots \\ &\approx F(u^-) + F'(u^-)(u-u^-) = \hat{F}(u)\,. \end{aligned}$$

The linear equation $\hat{F}(u) = 0$ has the solution

$$u = u^- - \frac{F(u^-)}{F'(u^-)}\,.$$

Expressed with an iteration index in the unknown, Newton's method t the more familiar mathematical form

$$u^{k+1} = u^k - \frac{F(u^k)}{F'(u^k)}, \quad k = 0, 1, \ldots$$

It can be shown that the error in iteration $k+1$ of Newton's method is the quare of the error in iteration k, a result referred to as *quadratic convergence*. his means that for small errors the method converges very fast, and in particular uch faster than Picard iteration and other iteration methods. (The proof of is result is found in most textbooks on numerical analysis.) However, the uadratic convergence appears only if u^k is sufficiently close to the solution. urther away from the solution the method can easily converge very slowly or iverge. The reader is encouraged to do Exercise 3 to get a better understanding r the behavior of the method.

Application of Newton's method to the logistic equation discretized by the ackward Euler method is straightforward as we have

$$F(u) = au^2 + bu + c, \quad a = \Delta t,\ b = 1 - \Delta t,\ c = -u^{(1)},$$

nd then

$$F'(u) = 2au + b\,.$$

he iteration method becomes

$$u = u^- + \frac{a(u^-)^2 + bu^- + c}{2au^- + b}, \quad u^- \ \leftarrow u\,. \tag{7}$$

t each time level, we start the iteration by setting $u^- = u^{(1)}$. Stopping criteria s listed for the Picard iteration can be used also for Newton's method.

An alternative mathematical form, where we write out a, b, and c, and use a me level counter n and an iteration counter k, takes the form

$$^{n,k+1} = u^{n,k} + \frac{\Delta t(u^{n,k})^2 + (1-\Delta t)u^{n,k} - u^{n-1}}{2\Delta t u^{n,k} + 1 - \Delta t}, \quad u^{n,0} = u^{n-1}, \quad k = 0, 1, \ldots \tag{8}$$

. program implementation is much closer to (7) than to (8), but the latter is etter aligned with the established mathematical notation used in the literature.

.7 Relaxation

ne iteration in Newton's method or Picard iteration consists of solving a linear roblem $\hat{F}(u) = 0$. Sometimes convergence problems arise because the new olution u of $\hat{F}(u) = 0$ is "too far away" from the previously computed solution $^-$. A remedy is to introduce a relaxation, meaning that we first solve $\hat{F}(u^*) = 0$ r a suggested value u^* and then we take u as a weighted mean of what we had, $^-$, and what our linearized equation $\hat{F} = 0$ suggests, u^*:

$$u = \omega u^* + (1-\omega)u^-\,.$$

The parameter ω is known as a *relaxation parameter*, and a choice $\omega <$ prevent divergent iterations.

Relaxation in Newton's method can be directly incorporated in tl iteration formula:

$$u = u^- - \omega\frac{F(u^-)}{F'(u^-)}\,.$$

1.8 Implementation and experiments

The program `logistic.py`[1] contains implementations of all the metl scribed above. Below is an extract of the file showing how the Picard and methods are implemented for a Backward Euler discretization of the equation.

```
def BE_logistic(u0, dt, Nt, choice='Picard',
                eps_r=1E-3, omega=1, max_iter=1000):
    if choice == 'Picard1':
        choice = 'Picard'
        max_iter = 1

    u = np.zeros(Nt+1)
    iterations = []
    u[0] = u0
    for n in range(1, Nt+1):
        a = dt
        b = 1 - dt
        c = -u[n-1]

        if choice == 'Picard':

            def F(u):
                return a*u**2 + b*u + c

            u_ = u[n-1]
            k = 0
            while abs(F(u_)) > eps_r and k < max_iter:
                u_ = omega*(-c/(a*u_ + b)) + (1-omega)*u_
                k += 1
            u[n] = u_
            iterations.append(k)

        elif choice == 'Newton':

            def F(u):
                return a*u**2 + b*u + c
<
            def dF(u):
                return 2*a*u + b

            u_ = u[n-1]
            k = 0
            while abs(F(u_)) > eps_r and k < max_iter:
                u_ = u_ - F(u_)/dF(u_)
```

[1] http://tinyurl.com/nm5587k/nonlin/logistic.py

```
            k += 1
        u[n] = u_
        iterations.append(k)
    return u, iterations
```

The Crank-Nicolson method utilizing a linearization based on the geometric ıean gives a simpler algorithm:

```
def CN_logistic(u0, dt, Nt):
    u = np.zeros(Nt+1)
    u[0] = u0
    for n in range(0, Nt):
        u[n+1] = (1 + 0.5*dt)/(1 + dt*u[n] - 0.5*dt)*u[n]
    return u
```

We may run experiments with the model problem (1) and the different ;rategies for dealing with nonlinearities as described above. For a quite coarse me resolution, $\Delta t = 0.9$, use of a tolerance $\epsilon_r = 0.1$ in the stopping criterion ıtroduces an iteration error, especially in the Picard iterations, that is visibly ıuch larger than the time discretization error due to a large Δt. This is lustrated by comparing the upper two plots in Figure 1. The one to the right as a stricter tolerance $\epsilon = 10^{-3}$, which leads to all the curves corresponding to icard and Newton iteration to be on top of each other (and no changes can be isually observed by reducing ϵ_r further). The reason why Newton's method does ıuch better than Picard iteration in the upper left plot is that Newton's method ith one step comes far below the ϵ_r tolerance, while the Picard iteration needs ı average 7 iterations to bring the residual down to $\epsilon_r = 10^{-1}$, which gives ısufficient accuracy in the solution of the nonlinear equation. It is obvious that ıe Picard1 method gives significant errors in addition to the time discretization nless the time step is as small as in the lower right plot.

The *BE exact* curve corresponds to using the exact solution of the quadratic quation at each time level, so this curve is only affected by the Backward Euler me discretization. The *CN gm* curve corresponds to the theoretically more ccurate Crank-Nicolson discretization, combined with a geometric mean for nearization. This curve appear as more accurate, especially if we take the plot ı the lower right with a small Δt and an appropriately small ϵ_r value as the xact curve.

When it comes to the need for iterations, Figure 2 displays the number of erations required at each time level for Newton's method and Picard iteration. he smaller Δt is, the better starting value we have for the iteration, and the ıster the convergence is. With $\Delta t = 0.9$ Picard iteration requires on average 2 iterations per time step, but this number is dramatically reduced as Δt is educed.

However, introducing relaxation and a parameter $\omega = 0.8$ immediately educes the average of 32 to 7, indicating that for the large $\Delta t = 0.9$, Picard eration takes too long steps. An approximately optimal value for ω in this ase is 0.5, which results in an average of only 2 iterations! Even more dramatic npact of ω appears when $\Delta t = 1$: Picard iteration does not convergence in 1000 erations, but $\omega = 0.5$ again brings the average number of iterations down to 2.

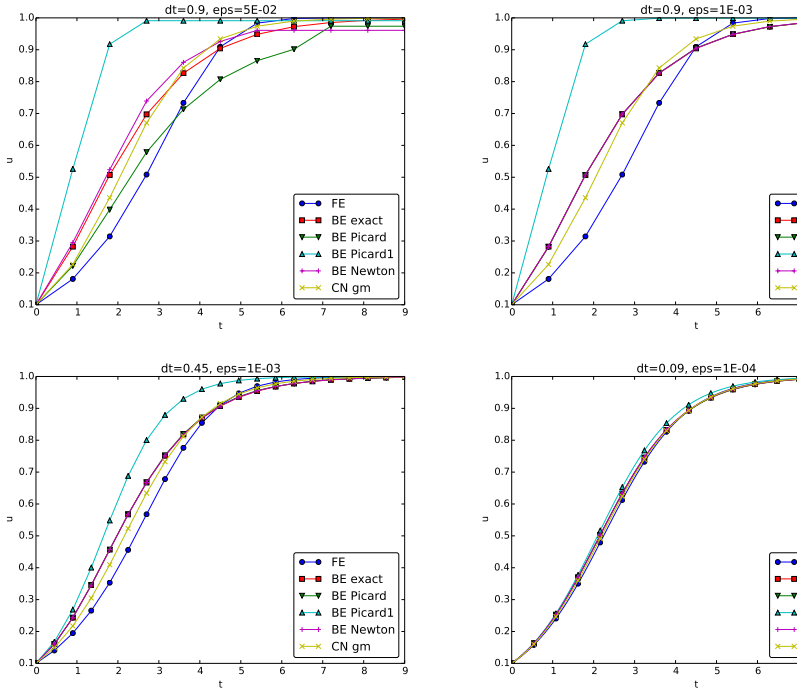
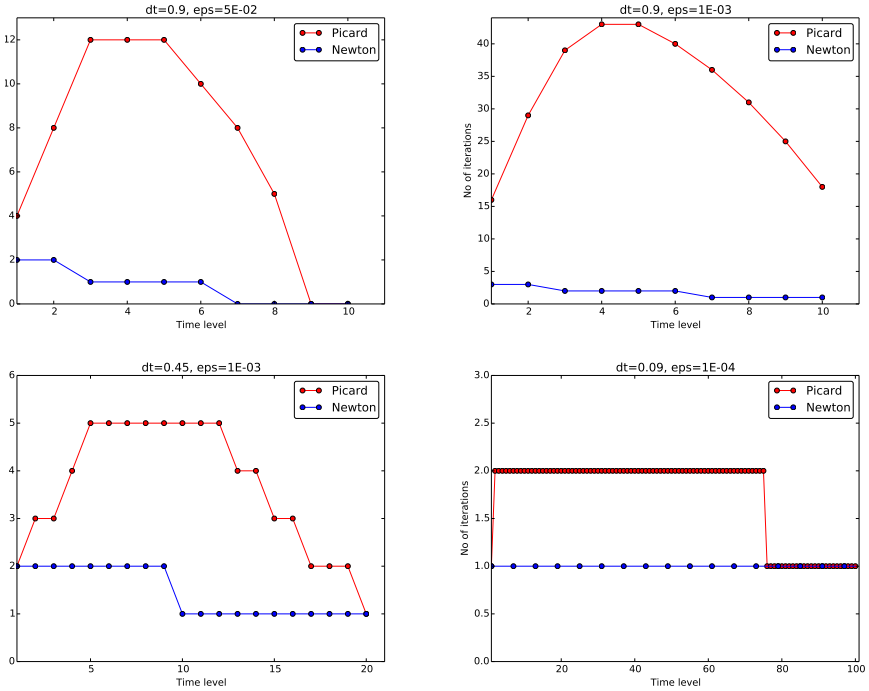


Figure 1: The impact of solution strategies and for four different ti lengths on the solution.

Experiments with this program reveal the relative performance of the ı as summarized in the table below. The Picard and Newton columns re typical number of iterations with these methods before the curve starts t out and the number of iterations is significantly reduced since the solutio nonlinear algebraic equation is very close to the starting value for the it (the solution at the previous time level). Increasing Δt moves the starti further away from the solution of the nonlinear equation and one exp increase in the number of iterations. Picard iteration is very much more s to the size of Δt than Newton's method. The tolerance ϵ_r in residu stopping criterion takes on a low and high value in the experiments.

Δt	ϵ_r	Picard	Newton
0.2	10^{-7}	5	2
0.2	10^{-3}	2	1
0.4	10^{-7}	12	3
0.4	10^{-3}	4	2
0.8	10^{-7}	58	3
0.8	10^{-3}	4	2



igure 2: Comparison of the number of iterations at various time levels for icard and Newton iteration.

emark. The simple Crank-Nicolson method with a geometric mean for the uadratic nonlinearity gives visually more accurate solutions than the Backward uler discretization. Even with a tolerance of $\epsilon_r = 10^{-3}$, all the methods for eating the nonlinearities in the Backward Euler discretization gives graphs that nnot be distinguished. So for accuracy in this problem, the time discretization much more crucial than ϵ_r. Ideally, one should estimate the error in the time iscretization, as the solution progresses, and set ϵ_r accordingly.

.9 Generalization to a general nonlinear ODE

et us see how the various methods in the previous sections can be applied to ie more generic model

$$u' = f(u, t), \tag{10}$$

here f is a nonlinear function of u.

xplicit time discretization. Explicit ODE methods like the Forward Euler cheme, Runge-Kutta methods, Adams-Bashforth methods all evaluate f at

time levels where u is already computed, so nonlinearities in f do not p difficulties.

Backward Euler discretization. Approximating u' by a backward di leads to a Backward Euler scheme, which can be written as

$$F(u^n) = u^n - \Delta t f(u^n, t_n) - u^{n-1} = 0,$$

or alternatively

$$F(u) = u - \Delta t f(u, t_n) - u^{(1)} = 0\,.$$

A simple Picard iteration, not knowing anything about the nonlinear st of f, must approximate $f(u, t_n)$ by $f(u^-, t_n)$:

$$\hat{F}(u) = u - \Delta t f(u^-, t_n) - u^{(1)}\,.$$

The iteration starts with $u^- = u^{(1)}$ and proceeds with repeating

$$u^* = \Delta t f(u^-, t_n) + u^{(1)}, \quad u = \omega u^* + (1-\omega)u^-, \quad u^- \leftarrow u,$$

until a stopping criterion is fulfilled.

Newton's method requires the computation of the derivative

$$F'(u) = 1 - \Delta t \frac{\partial f}{\partial u}(u, t_n)\,.$$

Starting with the solution at the previous time level, $u^- = u^{(1)}$, we can the standard formula

$$u = u^- - \omega \frac{F(u^-)}{F'(u^-)} = u^- - \omega \frac{u^{(1)} + \Delta t f(u^-, t_n)}{1 - \Delta t \frac{\partial}{\partial u} f(u^-, t_n)}\,.$$

The geometric mean trick cannot be used unless we know that f has a structure with quadratic expressions in u.

Crank-Nicolson discretization. The standard Crank-Nicolson sche arithmetic mean approximation of f takes the form

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{1}{2}(f(u^{n+1}, t_{n+1}) + f(u^n, t_n))\,.$$

We can write the scheme as a nonlinear algebraic equation

$$F(u) = u - u^{(1)} - \Delta t \frac{1}{2} f(u, t_{n+1}) - \Delta t \frac{1}{2} f(u^{(1)}, t_n) = 0\,.$$

A Picard iteration scheme must in general employ the linearization,

$$\hat{F}(u) = u - u^{(1)} - \Delta t \frac{1}{2} f(u^-, t_{n+1}) - \Delta t \frac{1}{2} f(u^{(1)}, t_n),$$

hile Newton's method can apply the general formula (11) with $F(u)$ given in .2) and

$$F'(u) = 1 - \frac{1}{2}\Delta t \frac{\partial f}{\partial u}(u, t_{n+1})\,.$$

Systems of nonlinear algebraic equations

nplicit time discretization methods for a system of ODEs, or a PDE, lead to *ystems* of nonlinear algebraic equations, written compactly as

$$F(u) = 0,$$

here u is a vector of unknowns $u = (u_0, \ldots, u_N)$, and F is a vector function: $= (F_0, \ldots, F_N)$. Sometimes the equation system has a special structure ecause of the underlying problem, e.g.,

$$A(u)u = b(u),$$

ith $A(u)$ as an $(N+1)\times(N+1)$ matrix function of u and b as a vector function: $= (b_0, \ldots, b_N)$.

We shall next explain how Picard iteration and Newton's method can be pplied to systems like $F(u) = 0$ and $A(u)u = b(u)$. The exposition has a focus n ideas and practical computations. More theoretical considerations, including uite general results on convergence properties of these methods, can be found ı Kelley [1].

.1 Picard iteration

Ve cannot apply Picard iteration to nonlinear equations unless there is some ɔecial structure. For the commonly arising case $A(u)u = b(u)$ we can linearize ıe product $A(u)u$ to $A(u^-)u$ and $b(u)$ as $b(u^-)$. That is, we use the most reviously computed approximation in A and b to arrive at a *linear system* for :

$$A(u^-)u = b(u^-)\,.$$

relaxed iteration takes the form

$$A(u^-)u^* = b(u^-), \quad u = \omega u^* + (1-\omega)u^-\,.$$

ı other words, we solve a system of nonlinear algebraic equations as a sequence f linear systems.

Algorithm for relaxed Picard iteration.

Given $A(u)u = b(u)$ and an initial guess u^-, iterate until convergence:

1. solve $A(u^-)u^* = b(u^-)$ with respect to u^*
2. $u = \omega u^* + (1-\omega)u^-$
3. $u^- \leftarrow u$

2.2 Newton's method

The natural starting point for Newton's method is the general nonlinea equation $F(u) = 0$. As for a scalar equation, the idea is to approximate F a known value u^- by a linear function $\hat{F}$, calculated from the first two t a Taylor expansion of F. In the multi-variate case these two terms bec

$$F(u^-) + J(u^-)\cdot(u - u^-),$$

where J is the *Jacobian* of F, defined by

$$J_{i,j} = \frac{\partial F_i}{\partial u_j}\,.$$

So, the original nonlinear system is approximated by

$$\hat{F}(u) = F(u^-) + J(u^-)\cdot(u - u^-) = 0,$$

which is linear in u and can be solved in a two-step procedure: fir $J\delta u = -F(u^-)$ with respect to the vector δu and then update $u = u^-$ relaxation parameter can easily be incorporated:

$$u = \omega(u^- + \delta u) + (1-\omega)u^- = u^- + \omega\delta u\,.$$

Algorithm for Newton's method.

Given $F(u) = 0$ and an initial guess u^-, iterate until convergence:

1. solve $J\delta u = -F(u^-)$ with respect to δu
2. $u = u^- + \omega\delta u$
3. $u^- \leftarrow u$

For the special system with structure $A(u)u = b(u)$,

$$F_i = \sum_k A_{i,k}(u)u_k - b_i(u),$$

ne gets

$$J_{i,j} = \sum_k \frac{\partial A_{i,k}}{\partial u_j} u_k + A_{i,j} - \frac{\partial b_i}{\partial u_j}\,. \tag{13}$$

Ve realize that the Jacobian needed in Newton's method consists of $A(u^-)$ as ı the Picard iteration plus two additional terms arising from the differentiation. sing the notation $A'(u)$ for $\partial A/\partial u$ (a quantity with three indices: $\partial A_{i,k}/\partial u_j$), nd $b'(u)$ for $\partial b/\partial u$ (a quantity with two indices: $\partial b_i/\partial u_j$), we can write the near system to be solved as

$$(A + A'u + b')\delta u = -Au + b,$$

r

$$(A(u^-) + A'(u^-)u^- + b'(u^-))\delta u = -A(u^-)u^- + b(u^-)\,.$$

earranging the terms demonstrates the difference from the system solved in ach Picard iteration:

$$\underbrace{A(u^-)(u^- + \delta u) - b(u^-)}_{\text{Picard system}} + \gamma(A'(u^-)u^- + b'(u^-))\delta u = 0\,.$$

ere we have inserted a parameter γ such that $\gamma = 0$ gives the Picard system nd $\gamma = 1$ gives the Newton system. Such a parameter can be handy in software o easily switch between the methods.

Combined algorithm for Picard and Newton iteration.

Given $A(u)$, $b(u)$, and an initial guess u^-, iterate until convergence:

1. solve $(A+\gamma(A'(u^-)u^-+b'(u^-)))\delta u = -A(u^-)u^-+b(u^-)$ with respect to δu
2. $u = u^- + \omega\delta u$
3. $u^- \leftarrow u$

$\gamma = 1$ gives a Newton method while $\gamma = 0$ corresponds to Picard iteration.

.3 Stopping criteria

et $||\cdot||$ be the standard Eucledian vector norm. Four termination criteria are uch in use:

- Absolute change in solution: $||u - u^-|| \leq \epsilon_u$

- Relative change in solution: $||u - u^-|| \leq \epsilon_u||u_0||$, where u_0 den start value of u^- in the iteration
- Absolute residual: $||F(u)|| \leq \epsilon_r$
- Relative residual: $||F(u)|| \leq \epsilon_r||F(u_0)||$

To prevent divergent iterations to run forever, one terminates the it when the current number of iterations k exceeds a maximum value k_{ma}

The relative criteria are most used since they are not sensitive to t acteristic size of u. Nevertheless, the relative criteria can be misleadir the initial start value for the iteration is very close to the solution, s unnecessary reduction in the error measure is enforced. In such cases the a criteria work better. It is common to combine the absolute and relative n of the size of the residual, as in

$$||F(u)|| \leq \epsilon_{rr}||F(u_0)|| + \epsilon_{ra},$$

where ϵ_{rr} is the tolerance in the relative criterion and ϵ_{ra} is the toleranc absolute criterion. With a very good initial guess for the iteration (typic solution of a differential equation at the previous time level), the term | is small and ϵ_{ra} is the dominating tolerance. Otherwise, $\epsilon_{rr}||F(u_0)||$ relative criterion dominates.

With the change in solution as criterion we can formulate a combined a and relative measure of the change in the solution:

$$||\delta u|| \leq \epsilon_{ur}||u_0|| + \epsilon_{ua},$$

The ultimate termination criterion, combining the residual and the in solution with a test on the maximum number of iterations allow, expressed as

$$||F(u)|| \leq \epsilon_{rr}||F(u_0)|| + \epsilon_{ra} \quad \text{or} \quad ||\delta u|| \leq \epsilon_{ur}||u_0|| + \epsilon_{ua} \quad \text{or} \quad k > k_{\text{ma}}$$

2.4 Example: A nonlinear ODE model from epidem

The simplest model spreading of a disease, such as a flu, takes the fo 2×2 ODE system

$$S' = -\beta SI,$$
$$I' = \beta SI - \nu I,$$

where $S(t)$ is the number of people who can get ill (susceptibles) and $I($ number of people who are ill (infected). The constants $\beta > 0$ and $\nu > 0$ given along with initial conditions $S(0)$ and $I(0)$.

mplicit time discretization. A Crank-Nicolson scheme leads to a 2×2 ystem of nonlinear algebraic equations in the unknowns S^{n+1} and I^{n+1}:

$$\frac{S^{n+1} - S^n}{\Delta t} = -\beta[SI]^{n+\frac{1}{2}} \approx -\frac{\beta}{2}(S^n I^n + S^{n+1} I^{n+1}), \tag{19}$$

$$\frac{I^{n+1} - I^n}{\Delta t} = \beta[SI]^{n+\frac{1}{2}} - \nu I^{n+\frac{1}{2}} \approx \frac{\beta}{2}(S^n I^n + S^{n+1} I^{n+1}) - \frac{\nu}{2}(I^n + I^{n+1})\,. \tag{20}$$

ntroducing S for S^{n+1}, $S^{(1)}$ for S^n, I for I^{n+1}, $I^{(1)}$ for I^n, we can rewrite the ystem as

$$F_S(S, I) = S - S^{(1)} + \frac{1}{2}\Delta t\beta(S^{(1)} I^{(1)} + SI) = 0, \tag{21}$$

$$F_I(S, I) = I - I^{(1)} - \frac{1}{2}\Delta t\beta(S^{(1)} I^{(1)} + SI) - \frac{1}{2}\Delta t\nu(I^{(1)} + I) = 0\,. \tag{22}$$

. Picard iteration. We assume that we have approximations S_- and I_- to and I. A way of linearizing the only nonlinear term SI is to write I_-S in ne $F_S = 0$ equation and S_-I in the $F_I = 0$ equation, which also *decouples* the quations. Solving the resulting linear equations with respect to the unknowns and I gives

$$S = \frac{S^{(1)} - \frac{1}{2}\Delta t\beta S^{(1)} I^{(1)}}{1 + \frac{1}{2}\Delta t\beta I_-},$$

$$I = \frac{I^{(1)} + \frac{1}{2}\Delta t\beta S^{(1)} I^{(1)}}{1 - \frac{1}{2}\Delta t\beta S_- + \nu}\,.$$

efore a new iteration, we must update $S^- \leftarrow S$ and $I^- \leftarrow I$.

lewton's method. The nonlinear system (21)-(22) can be written as $F(u) =$ with $F = (F_S, F_I)$ and $u = (S, I)$. The Jacobian becomes

$$J = \begin{pmatrix} \frac{\partial}{\partial S}F_S & \frac{\partial}{\partial I}F_S \\ \frac{\partial}{\partial S}F_I & \frac{\partial}{\partial I}F_I \end{pmatrix} = \begin{pmatrix} 1 + \frac{1}{2}\Delta t\beta I & \frac{1}{2}\Delta t\beta \\ -\frac{1}{2}\Delta t\beta S & 1 - \frac{1}{2}\Delta t\beta I - \frac{1}{2}\Delta t\nu \end{pmatrix}.$$

he Newton system $J(u^-)\delta u = -F(u^-)$ to be solved in each iteration is then

$$\begin{pmatrix} 1 + \frac{1}{2}\Delta t\beta I_- & \frac{1}{2}\Delta t\beta S_- \\ -\frac{1}{2}\Delta t\beta S_- & 1 - \frac{1}{2}\Delta t\beta I_- - \frac{1}{2}\Delta t\nu \end{pmatrix} \begin{pmatrix} \delta S \\ \delta I \end{pmatrix} = \begin{pmatrix} S_- - S^{(1)} + \frac{1}{2}\Delta t\beta(S^{(1)} I^{(1)} + S_- I_-) \\ I_- - I^{(1)} - \frac{1}{2}\Delta t\beta(S^{(1)} I^{(1)} + S_- I_-) - \frac{1}{2}\Delta t\nu(I^{(1)} + I_-) \end{pmatrix}$$

Remark. For this particular system of ODEs, explicit time integration m work very well. The 4-th order Runge-Kutta method is an excellent between high accuracy, high efficiency, and simplicity.

3 Linearization at the differential equation

The attention is now turned to nonlinear partial differential equations and application of the techniques explained above for ODEs. The model p is a nonlinear diffusion equation

$$\begin{aligned} \frac{\partial u}{\partial t} &= \nabla \cdot (\alpha(u)\nabla u) + f(u), && \boldsymbol{x} \in \Omega,\ t \in (0, T], \\ -\alpha(u)\frac{\partial u}{\partial n} &= g, && \boldsymbol{x} \in \partial\Omega_N,\ t \in (0, T], \\ u &= u_0, && \boldsymbol{x} \in \partial\Omega_D,\ t \in (0, T]\,. \end{aligned}$$

Our aim is to discretize the problem in time and then present tec for linearizing the time-discrete PDE problem "at the PDE level" su we transform the nonlinear stationary PDE problems at each time leve sequence of linear PDE problems, which can be solved using any met linear PDEs. This strategy avoids the solution systems of nonlinear a equations. In Section 4 we shall take the opposite (and more common) ap discretize the nonlinear problem in time and space first, and then so resulting nonlinear algebraic equations at each time level by the met Section 2.

3.1 Explicit time integration

The nonlinearities in the PDE are trivial to deal with if we choose an time integration method for (23), such as the Forward Euler method:

$$[D_t^+ u = \nabla \cdot (\alpha(u)\nabla u) + f(u)]^n,$$

or written out,

$$\frac{u^{n+1} - u^n}{\Delta t} = \nabla \cdot (\alpha(u^n)\nabla u^n) + f(u^n),$$

which is a linear equation in the unknown u^{n+1} with solution

$$u^{n+1} = u^n + \Delta t\nabla \cdot (\alpha(u^n)\nabla u^n) + \Delta t f(u^n)\,.$$

The disadvantage with this discretization is usually thought to be the s criterion

$$\Delta t \leq \frac{1}{\max \alpha}(\Delta x^2 + \Delta y^2 + \Delta z^2),$$

or the case $f = 0$ and a standard 2nd-order finite difference discretization in pace with mesh cell sizes Δx, Δy, and Δz in the various spatial directions.

.2 Backward Euler scheme and Picard iteration

Backward Euler scheme for (23) reads

$$[D_t^- u = \nabla \cdot (\alpha(u)\nabla u) + f(u)]^n \,.$$

Written out,

$$\frac{u^n - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^n)\nabla u^n) + f(u^n) \,. \tag{26}$$

This is a nonlinear, stationary PDE for the unknown function $u^n(\boldsymbol{x})$. We introduce a Picard iteration with k as iteration counter. A typical linearization of the $\nabla \cdot \alpha(u^n)\nabla u^n$ term in iteration $k+1$ is to use the previously computed $u^{n,k}$ approximation in the diffusion coefficient: $\alpha(u^{n,k})$. The nonlinear source term is treated similarly: $f(u^{n,k})$. The unknown function $u^{n,k+1}$ then fulfills the linear PDE

$$\frac{u^{n,k+1} - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k}) \,. \tag{27}$$

The initial guess for the Picard iteration at this time level can be taken as the solution at the previous time level: $u^{n,0} = u^{n-1}$.

We can alternatively apply the implementation-friendly notation where u corresponds to the unknown we want to solve for, i.e., $u^{n,k+1}$ above, and u^- is the most recently computed value, $u^{n,k}$ above. Moreover, $u^{(1)}$ denotes the known function at the previous time level, u^{n-1} above. The PDE to be solved in a Picard iteration then looks like

$$\frac{u - u^{(1)}}{\Delta t} = \nabla \cdot (\alpha(u^-)\nabla u) + f(u^-) \,. \tag{28}$$

At the beginning of the iteration we start with the value from the previous time level: $u^- = u^{(1)}$, and after each iteration, u^- is updated to u.

.3 Backward Euler scheme and Newton's method

At time level n we have to solve the stationary PDE (26), this time with Newton's method. Normally, Newton's method is defined for systems of *algebraic equations*, but the idea of the method can be applied at the PDE level too.

Linearization via Taylor expansions. Let $u^{n,k}$ be an approximation to the unknown u^n. We seek a better approximation on the form

$$u^n = u^{n,k} + \delta u \,. \tag{29}$$

The idea is to insert (29) in (26), Taylor expand the nonlinearities a keep the terms that are linear in δu. Then we can solve a linear PDE correction δu and use (29) to find a new approximation $u^{n,k+1} = u^{n,k}$ u^n.

Inserting (29) in (26) gives

$$\frac{u^{n,k} + \delta u - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^{n,k} + \delta u)\nabla(u^{n,k} + \delta u)) + f(u^{n,k} + \delta u)$$

We can Taylor expand $\alpha(u^{n,k} + \delta u)$ and $f(u^{n,k} + \delta u)$:

$$\alpha(u^{n,k} + \delta u) = \alpha(u^{n,k}) + \frac{d\alpha}{du}(u^{n,k})\delta u + \mathcal{O}(\delta u^2) \approx \alpha(u^{n,k}) + \alpha'(u^{n,k}$$
$$f(u^{n,k} + \delta u) = f(u^{n,k}) + \frac{df}{du}(u^{n,k})\delta u + \mathcal{O}(\delta u^2) \approx f(u^{n,k}) + f'(u^{n,k}$$

Inserting the linear approximations of α and f in (30) results in

$$\begin{aligned}\frac{u^{n,k} + \delta u - u^{n-1}}{\Delta t} &= \nabla \cdot (\alpha(u^{n,k})\nabla u^{n,k}) + f(u^{m,k}) + \\ &\nabla \cdot (\alpha(u^{n,k})\nabla \delta u) + \nabla \cdot (\alpha'(u^{n,k})\delta u \nabla u^{n,k}) + \\ &\nabla \cdot (\alpha'(u^{n,k})\delta u \nabla \delta u) + f'(u^{n,k})\delta u\end{aligned}$$

The term $\alpha'(u^{n,k})\delta u \nabla \delta u$ is $\mathcal{O}(\delta u^2)$ and therefore omitted. Reorganiz equation gives a PDE for δu that we can write in short form as

$$\delta F(\delta u; u^{n,k}) = -F(u^{n,k}),$$

where

$$\begin{aligned}F(u^{n,k}) &= \frac{u^{n,k} - u^{n-1}}{\Delta t} - \nabla \cdot (\alpha(u^{n,k})\nabla u^{n,k}) + f(u^{n,k}), \\ \delta F(\delta u; u^{n,k}) &= -\frac{1}{\Delta t}\delta u + \nabla \cdot (\alpha(u^{n,k})\nabla \delta u) + \\ &\nabla \cdot (\alpha'(u^{n,k})\delta u \nabla u^{n,k}) + f'(u^{n,k})\delta u \,.\end{aligned}$$

Note that δF is a linear function of δu, and F contains only terms t known, such that the PDE for δu is indeed linear.

Observations.

The notational form $\delta F = -F$ resembles the Newton system $J\delta u = -$ systems of algebraic equations, with δF as $J\delta u$. The unknown vecto linear system of algebraic equations enters the system as a linear ope

in terms of a matrix-vector product ($J\delta u$), while at the PDE level we have a linear differential operator instead (δF).

imilarity with Picard iteration. We can rewrite the PDE for δu in a ightly different way too if we define $u^{n,k} + \delta u$ as $u^{n,k+1}$.

$$\frac{u^{n,k+1} - u^{n-1}}{\Delta t} = \nabla\cdot(\alpha(u^{n,k})\nabla u^{n,k+1}) + f(u^{n,k}) + \nabla\cdot(\alpha'(u^{n,k})\delta u\nabla u^{n,k}) + f'(u^{n,k})\delta u\,. \tag{34}$$

ote that the first line is the same PDE as arise in the Picard iteration, while the emaining terms arise from the differentiations that are an inherent ingredient ı Newton's method.

mplementation. For coding we want to introduce u for u^n, u^- for $u^{n,k}$ and $^{(1)}$ for u^{n-1}. The formulas for F and δF are then more clearly written as

$$F(u^-) = \frac{u^- - u^{(1)}}{\Delta t} - \nabla\cdot(\alpha(u^-)\nabla u^-) + f(u^-), \tag{35}$$

$$\delta F(\delta u; u^-) = -\frac{1}{\Delta t}\delta u + \nabla\cdot(\alpha(u^-)\nabla\delta u) + \nabla\cdot(\alpha'(u^-)\delta u\nabla u^-) + f'(u^-)\delta u\,. \tag{36}$$

he form that orders the PDE as the Picard iteration terms plus the Newton ıethod's derivative terms becomes

$$\frac{u - u^{(1)}}{\Delta t} = \nabla\cdot(\alpha(u^-)\nabla u) + f(u^-) + \gamma(\nabla\cdot(\alpha'(u^-)(u-u^-)\nabla u^-) + f'(u^-)(u-u^-))\,. \tag{37}$$

he Picard and full Newton versions correspond to $\gamma = 0$ and $\gamma = 1$, respectively.

.4 Crank-Nicolson discretization

Crank-Nicolson discretization of (23) applies a centered difference at $t_{n+\frac{1}{2}}$:

$$[D_t u = \nabla\cdot(\alpha(u)\nabla u) + f(u)]^{n+\frac{1}{2}}\,.$$

ince u is not known at $t_{n+\frac{1}{2}}$ we need to express the terms on the right-hand side ia unknowns u^n and u^{n+1}. The standard technique is to apply an arithmetic verage,

$$u^{n+\frac{1}{2}} \approx \frac{1}{2}(u^n + u^{n+1})\,.$$

However, with nonlinear terms we have many choices of formulating an ari mean:

$$[f(u)]^{n+\frac{1}{2}} \approx f(\frac{1}{2}(u^n + u^{n+1})) = [f(\overline{u}^t)]^{n+\frac{1}{2}},$$

$$[f(u)]^{n+\frac{1}{2}} \approx frac12(f(u^n) + f(u^{n+1})) = [\overline{f(u)}^t]^{n+\frac{1}{2}},$$

$$[\alpha(u)\nabla u]^{n+\frac{1}{2}} \approx \alpha(\frac{1}{2}(u^n + u^{n+1}))\nabla(\frac{1}{2}(u^n + u^{n+1})) = \alpha(\overline{u}^t)\nabla\overline{u}^t]^{n+\frac{1}{2}},$$

$$[\alpha(u)\nabla u]^{n+\frac{1}{2}} \approx \frac{1}{2}(\alpha(u^n) + \alpha(u^{n+1}))\nabla(\frac{1}{2}(u^n + u^{n+1})) = [\overline{\alpha(u)}^t\nabla\overline{u}^t]^{n+}$$

$$[\alpha(u)\nabla u]^{n+\frac{1}{2}} \approx \frac{1}{2}(\alpha(u^n)\nabla u^n + \alpha(u^{n+1})\nabla u^{n+1}) = [\overline{\alpha(u)\nabla u}^t]^{n+\frac{1}{2}}\,.$$

4 Discretization of stationary nonlinear diff tial equations

Section 3 presents methods for linearizing time-discrete PDEs directly discretization in space. We can alternatively carry out the discretization and of the time-discrete nonlinear PDE problem and get a system of n algebraic equations, which can be solved by Picard iteration or Newton's as presented in Section 2. This latter approach will now be described i

We shall work with the 1D problem

$$-(\alpha(u)u')' + au = f(u), \quad x\in(0,L), \quad \alpha(u(0))u'(0) = C,\ u(L) = L$$

This problem is of the same nature as those arising from implicit time int of a nonlinear diffusion PDE as outlined in Section 3.2 (set $a = 1/\Delta t$ $f(u)$ incorporate the nonlinear source term as well as known terms v time-dependent unknown function at the previous time level).

4.1 Finite difference discretizations

The nonlinearity in the differential equation (43) poses no more difficulty variable coefficient, as in $(\alpha(x)u')'$. We can therefore use a standard a to discretizing the Laplace term with a variable coefficient:

$$[-D_x\alpha D_x u + au = f]_i\,.$$

Writing this out for a uniform mesh with points $x_i = i\Delta x$, $i = 0,\ldots,N$ to

$$-\frac{1}{\Delta x^2}\left(\alpha_{i+\frac{1}{2}}(u_{i+1} - u_i) - \alpha_{i-\frac{1}{2}}(u_i - u_{i-1})\right) + au_i = f(u_i)\,.$$

his equation is valid at all the mesh points $i = 0, 1, \ldots, N_x - 1$. At $i = N_x$ e have the Dirichlet condition $u_i = 0$. The only difference from the case with $\alpha(x)u')'$ and $f(x)$ is that now α and f are functions of u and not only on x: $\alpha(u(x))u')'$ and $f(u(x))$.

The quantity $\alpha_{i+\frac{1}{2}}$, evaluated between two mesh points, needs a comment. ince α depends on u and u is only known at the mesh points, we need to express $_{i+\frac{1}{2}}$ in terms of u_i and u_{i+1}. For this purpose we use an arithmetic mean, lthough a harmonic mean is also common in this context if α features large umps. There are two choices of arithmetic means:

$$\alpha_{i+\frac{1}{2}} \approx \alpha(\frac{1}{2}(u_i + u_{i+1}) = [\alpha(\overline{u}^x)]^{i+\frac{1}{2}}, \tag{45}$$

$$\alpha_{i+\frac{1}{2}} \approx \frac{1}{2}(\alpha(u_i) + \alpha(u_{i+1})) = [\overline{\alpha(u)}^x]^{i+\frac{1}{2}} \tag{46}$$

quation (44) with the latter approximation then looks like

$$-\frac{1}{2\Delta x^2}\left((\alpha(u_i) + \alpha(u_{i+1}))(u_{i+1} - u_i) - (\alpha(u_{i-1}) + \alpha(u_i))(u_i - u_{i-1})\right) + au_i = f(u_i), \tag{47}$$

r written more compactly,

$$[-D_x\overline{\alpha}^x D_x u + au = f]_i\,.$$

At mesh point $i = 0$ we have the boundary condition $\alpha(u)u' = C$, which is iscretized by

$$[\alpha(u)D_{2x}u = C]_0,$$

ıeaning

$$\alpha(u_0)\frac{u_1 - u_{-1}}{2\Delta x} = C\,. \tag{48}$$

he fictitious value u_{-1} can be eliminated with the aid of (47) for $i = 0$. Formally, 47) should be solved with respect to u_{i-1} and that value (for $i = 0$) should be ıserted in (48), but it is algebraically much easier to do it the other way around. lternatively, one can use a ghost cell $[-\Delta x, 0]$ and update the u_{-1} value in ıe ghost cell according to (48) after every Picard or Newton iteration. Such n approach means that we use a known u_{-1} value in (47) from the previous eration.

.2 Solution of algebraic equations

'he structure of the equation system. The nonlinear algebraic equations 47) are of the form $A(u)u = b(u)$ with

$$A_{i,i} = \frac{1}{2\Delta x^2}(-\alpha(u_{i-1}) + 2\alpha(u_i) - \alpha(u_{i+1})) + a,$$
$$A_{i,i-1} = -\frac{1}{2\Delta x^2}(\alpha(u_{i-1}) + \alpha(u_i)),$$
$$A_{i,i+1} = -\frac{1}{2\Delta x^2}(\alpha(u_i) + \alpha(u_{i+1})),$$
$$b_i = f(u_i)\,.$$

The matrix $A(u)$ is tridiagonal: $A_{i,j} = 0$ for $j > 1 + 1$ and $j < i - 1$.

The above expressions are valid for internal mesh points $1 \le i \le N_x$ $i = 0$ we need to express $u_{i-1} = u_{-1}$ in terms of u_1 using (48):

$$u_{-1} = u_1 - \frac{2\Delta x}{\alpha(u_0)}\,.$$

This value must be inserted in $A_{0,0}$. The expression for $A_{i,i+1}$ applies f and $A_{i,i-1}$ does not enter the system when $i = 0$.

Regarding the last equation, its form depends on whether we incl Dirichlet condition $u(L) = D$, meaning $u_{N_x} = D$, in the nonlinear a equation system or not. Suppose we choose $(u_0, u_1, \ldots, u_{N_x-1})$ as un later referred to as *equations without Dirichlet conditions*. The last equa responds to $i = N_x - 1$. It involves the boundary value u_{N_x}, which is sub by D. If the unknown vector includes the boundary value, $(u_0, u_1, .$ later referred to as *equations including Dirichlet conditions*, the equa $i = N_x - 1$ just involves the unknown u_{N_x}, and the final equation l $u_{N_x} = D$, corresponding to $A_{i,i} = 1$ and $b_i = D$ for $i = N_x$.

Picard iteration. The obvious Picard iteration scheme is to use pr computed values of u_i in $A(u)$ and $b(u)$, as described more in detail tion 2. The system $F(u) = 0$ is then solved with respect to u, whe $(F_0, F_1, \ldots, F_m)$, $u = (u_0, u_1, \ldots, u_m)$, and the F_i expression is given abo index m is N_x in equations including the Dirichlet condition and $N_x -$ the Dirichlet condition is excluded.

To write out the mathematical details, we introduce u^- as the mos approximation to solution vector u, and u_i^- is the i-th component in u, the most recently computed value of the unknown u_i. For the case N_x get the following system to solve in case we omit the Dirichlet conditi the system:

$$\begin{pmatrix} \frac{1}{2\Delta x^2}(-\alpha(u_1^-) + 2\alpha(u_0^-) - \alpha(u_1^-)) + a & -\frac{1}{2\Delta x^2}(\alpha(u_0^-) + \alpha(u_1^- \\ -\frac{1}{2\Delta x^2}(\alpha(u_0^-) + \alpha(u_1^-)) & \frac{1}{2\Delta x^2}(-\alpha(u_0^-) + 2\alpha(u_1^-) - \alpha(\end{pmatrix}$$

where u_{-1} must be substituted by (49), and u_2 by D.

The system with the Dirichlet condition becomes

$$
\begin{array}{ccc}
\frac{1}{2\Delta x^2}(-\alpha(u_1^-)+2\alpha(u_0^-)-\alpha(u_1^-))+a & 0 & \\
-\frac{1}{2\Delta x^2}(\alpha(u_0^-)+\alpha(u_1^-)) & & \\
-\frac{1}{2\Delta x^2}(\alpha(u_0^-)+\alpha(u_1^-)) & \frac{1}{2\Delta x^2}(-\alpha(u_0^-)+2\alpha(u_1^-)-\alpha(u_2))+a & - \\
0 & 0 &
\end{array}
$$

Newton's method. The Jacobian must be derived in order to use Newton's method. Here it means that we need to differentiate $F(u) = A(u)u - b(u)$ with respect to the unknown parameters $u_0, u_1, \ldots, u_m$ ($m = N_x$ or $m = N_x - 1$, depending on whether the Dirichlet condition is included in the nonlinear system $F(u) = 0$ or not). Nonlinear equation number i has the structure

$$
F_i = A_{i,i-1}(u_{i-1}, u_i)u_{i-1} + A_{i,i}(u_{i-1}, u_i, u_{i+1})u_i + A_{i,i+1}(u_i, u_{i+1})u_{i+1} - b_i(u_i)\,.
$$

Computing the Jacobian requires careful differentiation. For example,

$$
\begin{aligned}
\frac{\partial}{\partial u_i}(A_{i,i}(u_{i-1}, u_i, u_{i+1})u_i) &= \frac{\partial A_{i,i}}{\partial u_i}u_i + A_{i,i}\frac{\partial u_i}{\partial u_i} \\
&= \frac{\partial}{\partial u_i}(\frac{1}{2\Delta x^2}(-\alpha(u_{i-1}) + 2\alpha(u_i) - \alpha(u_{i+1})) + a)u_i + \\
&\quad \frac{1}{2\Delta x^2}(-\alpha(u_{i-1}) + 2\alpha(u_i) - \alpha(u_{i+1})) + a) \\
&= \frac{1}{2\Delta x^2}(2\alpha'(u_i)u_i - \alpha(u_{i-1}) + 2\alpha(u_i) - \alpha(u_{i+1})) + a\,.
\end{aligned}
$$

The complete Jacobian becomes

$$
\begin{aligned}
J_{i,i} &= \frac{\partial F_i}{\partial u_i} = \frac{\partial A_{i,i-1}}{\partial u_i}u_{i-1} + \frac{\partial A_{i,i}}{\partial u_i}u_i + A_{i,i} + \frac{\partial A_{i,i+1}}{\partial u_i}u_{i+1} - \frac{\partial b_i}{\partial u_i} \\
&= \frac{1}{2\Delta x^2}(-\alpha'(u_i)u_{i-1} + 2\alpha'(u_i)u_i - \alpha(u_{i-1}) + 2\alpha(u_i) - \alpha(u_{i+1})) + \\
&\quad a - \frac{1}{2\Delta x^2}\alpha'(u_i)u_{i+1} - b'(u_i), \\
J_{i,i-1} &= \frac{\partial F_i}{\partial u_{i-1}} = \frac{\partial A_{i,i-1}}{\partial u_{i-1}}u_{i-1} + A_{i-1,i} + \frac{\partial A_{i,i}}{\partial u_{i-1}}u_i - \frac{\partial b_i}{\partial u_{i-1}} \\
&= \frac{1}{2\Delta x^2}(-\alpha'(u_{i-1})u_{i-1} - (\alpha(u_{i-1}) + \alpha(u_i)) + \alpha'(u_{i-1})u_i), \\
J_{i,i+1} &= \frac{\partial A_{i,i+1}}{\partial u_{i-1}}u_{i+1} + A_{i+1,i} + \frac{\partial A_{i,i}}{\partial u_{i+1}}u_i - \frac{\partial b_i}{\partial u_{i+1}} \\
&= \frac{1}{2\Delta x^2}(-\alpha'(u_{i+1})u_{i+1} - (\alpha(u_i) + \alpha(u_{i+1})) + \alpha'(u_{i+1})u_i)\,.
\end{aligned}
$$

The explicit expression for nonlinear equation number i, $F_i(u_0, u_1, \ldots)$, arises from moving the (u_i) term in (47) to the left-hand side:

$$
\begin{aligned}
F_i = & = -\frac{1}{2\Delta x^2}((\alpha(u_i) + \alpha(u_{i+1}))(u_{i+1} - u \\
& + au_i - f(u_i) = 0\,.
\end{aligned}
$$

At the boundary point $i = 0$, u_{-1} must be replaced using the form
When the Dirichlet condition at $i = N_x$ is not a part of the equation
the last equation $F_m = 0$ for $m = N_x - 1$ involves the quantity u_{N_x}
must be replaced by D. If u_{N_x} is treated as an unknown in the system,
equation $F_m = 0$ has $m = N_x$ and reads

$$
F_{N_x}(u_0, \ldots, u_{N_x}) = u_{N_x} - D = 0\,.
$$

Similar replacement of u_{-1} and u_{N_x} must be done in the Jacobian for
and last row. When u_{N_x} is included as an unknown, the last row in the J
must help implement the condition $\delta u_{N_x} = 0$, since we assume that u
the right Dirichlet value at the beginning of the iteration ($u_{N_x} = D$), a
the Newton update should be zero for $i = 0$, i.e., $\delta u_{N_x} = 0$. This also fo
right-hand side to be $b_i = 0$, $i = N_x$.

We have seen, and can see from the present example, that the linear
in Newton's method contains all the terms present in the system tha
in the Picard iteration method. The extra terms in Newton's method
multiplied by a factor such that it is easy to program one linear system
this factor to 0 or 1 to generate the Picard or Newton system.

4.3 Galerkin-type discretizations

For the finite element discretization we first need to derive the var
problem. Let V be an appropriate function space with basis functions {
Because of the Dirichlet condition at $x = L$ we require $\psi_i(L) = 0$, $i \in$
approximate solution is written as $u = D + \sum_{j\in\mathcal{I}_s} c_j\psi_j$, where the tern
be viewed as a boundary function needed to implement the Dirichlet cc
$u(L) = D$.

Using Galerkin's method, we multiply the differential equation by an
and integrate terms with second-order derivatives by parts:

$$
\int_0^L \alpha(u)u'v'\,\mathrm{d}x + \int_0^L auv\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x + [\alpha(u)u'v]_0^L, \quad \forall v \in
$$

The Neumann condition at the boundary $x = 0$ is inserted in the bounda

$$
[\alpha(u)u'v]_0^L = \alpha(u(L))u'(L)v(L) - \alpha(u(0))u'(0)v(0) = 0 - Cv(0) = -C
$$

(Recall that since $\psi_i(L) = 0$, any linear combination v of the basis functi
vanishes at $x = L$: $v(L) = 0$.) The variational problem is then: find $u \in$
that

$$\int_0^L \alpha(u)u'v'\,\mathrm{d}x + \int_0^L auv\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x - Cv(0), \quad \forall v \in V\,. \tag{51}$$

To derive the algebraic equations, we note that $\forall v \in V$ is equivalent with $= \psi_i$ for $i \in \mathcal{I}_s$. Setting $u = D + \sum_j c_j\psi_j$ and sorting terms results in the near system

$$\sum_{\in\mathcal{I}_s}\left(\int_0^L \alpha(D + \sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_j'\psi_i'\,\mathrm{d}x\right)c_j = \int_0^L f(D+\sum_{k\in\mathcal{I}_s} c_k\psi_k)\psi_i\,\mathrm{d}x - C\psi_i(0), \quad i \in \tag{52}$$

undamental integration problem. Methods that use the Galerkin or eighted residual principle face a fundamental difficulty in nonlinear problems: ow can we integrate a terms like $\int_0^L \alpha(\sum_k c_k\psi_k)\psi_i'\psi_j'\,\mathrm{d}x$ and $\int_0^L f(\sum_k c_k\psi_k)\psi_i\,\mathrm{d}x$ hen we do not know the c_k coefficients in the argument of the α function? We an resort to numerical integration, provided an approximate $\sum_k c_k\psi_k$ can be sed for the argument u in f and α. If we want to derive the structure of the onlinear algebraic equations, we need to apply numerical integration based on ıe nodes only and/or the group finite element method.

.4 Finite element basis functions

ıtroduction of finite element basis functions φ_i means setting

$$\psi_i = \varphi_{\nu(i)}, \quad i \in \mathcal{I}_s,$$

here degree of freedom number $\nu(i)$ in the mesh corresponds to unknown umber i (c_i). The expansion of u can still be

$$u = D + \sum_{j\in\mathcal{I}_s} c_j\varphi_{\nu(j)},$$

ut is more common in a finite element context to use a boundary function $= \sum_{j\in I_b} U_j\varphi_j$, where U_j are prescribed Dirichlet conditions for degree of eedom number j and U_j is the corresponding value. In the present example, ıis means

$$u = D\varphi_0 + \sum_{j\in\mathcal{I}_s} c_j\varphi_{j+1}, \quad \mathcal{I}_s = \{0,\ldots,N_n-2\}\,.$$

ı the general case with u prescribed as U_j at some nodes $j \in I_b$, we set

$$u = \sum_{j\in I_b} U_j\varphi_j + \sum_{j\in\mathcal{I}_s} c_j\varphi_{\nu(j)},$$

here $c_j = u(x^{\nu(j)})$. That is, $\nu(j)$ maps unknown number j to the corresponding ode number $\nu(j)$ such that $c_j = u(x^{\nu(j)})$.

4.5 The group finite element method

Finite element approximation of functions of u. Since we already u as $\sum_j \varphi_j u(x_j)$, we may use the same approximation for other func well. For example,

$$f(u) \approx \sum_j f(x_j)\varphi_j,$$

where $f(x_j)$ is the value of f at node j. Since f is a function of u, $f(u(x_j))$. Introducing u_j as a short form for $u(x_j)$, we can write

$$f(u) \approx \sum_j f(u_j)\varphi_j\,.$$

This approximation is known as the *group finite element method* or the *approximation* technique. The index j runs over all node numbers in th

The principal advantages of the group finite element method are tw

1. Complicated nonlinear expressions can be simplified to increase ciency of numerical computations.
2. One can derive *symbolic form* of the difference equations arising f finite element method in nonlinear problems. The symbolic form i for comparing finite element and finite difference equations of nc differential equation problems.

Below, we shall explore point 2 to see exactly how the finite element creates more complex expressions in the resulting linear system (the di equations) that the finite difference method does. It turns out that is very to see what kind of turns in the difference equations that arise from $\int f($ without using the group finite element method or numerical integration the nodes only.

Note, however, that an expression like $\int f(u)\varphi_i\,\mathrm{d}x$ causes no proble computer program as the integral is calculated by numerical integratic an existing approximation of u in $f(u)$ such that the integrand can be s at any spatial point.

Simplified problem. Our aim now is the derive symbolic expressions difference equations arising from the finite element method in nonlinear p and compare the expressions with those arising in the finite difference To this, let us simplify the model problem and set $a = 0$, $\alpha = 1$, $f(u) =$ have Neumann conditions at both ends such that we get a very simple n problem $-u'' = u^2$, $u'(0) = 1$, $u'(L) = 0$. The variational form is then

$$\int_0^L u'v'\,\mathrm{d}x = \int_0^L u^2v\,\mathrm{d}x - v(0), \quad \forall v \in V\,.$$

The term with $u'v'$ is well known so the only new feature is the term $\int$

To make the distance from finite element equations to finite difference equa-
ons as short as possible, we shall substitute c_j in the sum $u = \sum_j c_j\varphi_j$ by
$_j = u(x_j)$ since c_j is the value of u at node j. (In the more general case with
irichlet conditions as well, we have a sum $\sum_j c_j\varphi_{\nu(j)}$ where c_j is replaced by
$(x_{\nu(j)})$. We can then introduce some other counter k such that it is meaningful
write $u = \sum_k u_k\varphi_k$, where k runs over appropriate node numbers.) The
uantity u_j in $\sum_j u_j\varphi_j$ is the same as u at mesh point number j in the finite
ifference method, which is commonly denoted u_j.

ntegrating nonlinear functions. Consider the term $\int u^2 v\,\mathrm{d}x$ in the varia-
onal formulation with $v = \varphi_i$ and $u = \sum_k \varphi_k u_k$:

$$\int_0^L (\sum_k u_k\varphi_k)^2\varphi_i\,\mathrm{d}x\,.$$

valuating this integral for P1 elements (see Problem 11) results in the expression

$$\frac{h}{12}(u_{i-1}^2 + 2u_i(u_{i-1}+u_{i+1}) + 6u_i^2 + u_{i+1}^2),$$

be compared with the simple value u_i^2 that would arise in a finite difference
iscretization when u^2 is sampled at mesh point x_i. More complicated $f(u)$
nctions give rise to much more lengthy expressions, if it is possible to carry
ut the integral symbolically at all.

pplication of the group finite element method. Let use the group finite
ement method to derive the terms in the difference equation corresponding to
(u) in the differential equation. We have

$$\int_0^L f(u)\varphi_i\,\mathrm{d}x \approx \int_0^L (\sum_j \varphi_j f(u_j))\varphi_i\,\mathrm{d}x = \sum_j \left(\int_0^L \varphi_i\varphi_j\,\mathrm{d}x\right) f(u_j)\,.$$

Ve recognize this expression as the mass matrix M, arising from $\int \varphi_i\varphi_j\,\mathrm{d}x$,
mes the vector $f = (f(u_0), f(u_1), \ldots,)$: Mf. The associated terms in the
ifference equations are, for P1 elements,

$$\frac{h}{6}(f(u_{i-1}) + 4f(u_i) + f(u_{i+1}))\,.$$

ccasionally, we want to interpret this expression in terms of finite differences,
nd to this end a rewrite of this expression is convenient:

$$\frac{h}{6}(f(u_{i-1}) + 4f(u_i) + f(u_{i+1})) = h[f(u) - \frac{h^2}{6}D_xD_xf(u)]_i\,.$$

hat is, the finite element treatment of $f(u)$ (when using a group finite element
ethod) gives the same term as in a finite difference approach, $f(u_i)$, minus a
iffusion term which is the 2nd-order discretization of $\frac{1}{6}h^2f''(x_i)$.

We may lump the mass matrix through integration with the Trapezoi
so that M becomes diagonal in the finite element method. In that case
term in the differential equation gives rise to a single term $hf(u_i)$, just a
finite difference method.

4.6 Numerical integration of nonlinear terms

Let us reconsider a term $\int f(u)v\,\mathrm{d}x$ as treated in the previous secti
now we want to integrate this term numerically. Such an approach can
easy-to-interpret formulas if we apply a numerical integration rule that
the integrand at the node points x_i only, because at such points, $\varphi_j(x_
$j \neq i$, which leads to great simplifications.

The term in question takes the form

$$\int_0^L f(\sum_k u_k\varphi_k)\varphi_i\,\mathrm{d}x\,.$$

Evaluation of the integrand at a node x_ℓ leads to a collapse of the sum $\sum$
to one term because

$$\sum_k u_k\varphi_k(x_\ell) = u_\ell\,.$$

$$f(\sum_k u_k \underbrace{\varphi_k(x_\ell)}_{\delta_{k\ell}})\underbrace{\varphi_i(x_\ell)}_{\delta_{i\ell}} = f(u_\ell)\delta_{i\ell},$$

where we have used the Kronecker delta: $\delta_{ij} = 0$ if $i \neq j$ and $\delta_{ij} = 1$ if

Considering the Trapezoidal rule for integration, where the integratio
are the nodes, we have

$$\int_0^L f(\sum_k u_k\varphi_k)(x)\varphi_i(x)\,\mathrm{d}x \approx h\sum_{\ell=0}^{N_n} f(u_\ell)\delta_{i\ell} - \mathcal{C} = hf(u_i)\,.$$

This is the same representation of the f term as in the finite difference
The term $\mathcal{C}$ contains the evaluations of the integrand at the ends with w
needed to make a true Trapezoidal rule:

$$\mathcal{C} = \frac{h}{2}f(u_0)\varphi_i(0) + \frac{h}{2}f(u_{N_n-1})\varphi_i(L)\,.$$

The answer $hf(u_i)$ must therefore be multiplied by $\frac{1}{2}$ if $i = 0$ or $i =$
Note that $\mathcal{C} = 0$ for $i = 1, \ldots, N_n - 2$.

One can alternatively use the Trapezoidal rule on the reference
assemble the contributions. It is a bit more labor in this context, but wo
the reference cell is safer as that approach is guaranteed to handle disco
derivatives of finite element functions correctly (not important in this pa
example), while the rule above was derived with the assumption th
continuous at the integration points.

The conclusion is that it suffices to use the Trapezoidal rule if one wants
o derive the difference equations in the finite element method and make them
milar to those arising in the finite difference method. The Trapezoidal rule
as sufficient accuracy for P1 elements, but for P2 elements one should turn to
impson's rule.

.7 Finite element discretization of a variable coefficient Laplace term

urning back to the model problem (43), it remains to calculate the contribution
f the $(\alpha u')'$ and boundary terms to the difference equations. The integral in
ie variational form corresponding to $(\alpha u')'$ is

$$\int_0^L \alpha(\sum_k c_k\psi_k)\psi_i'\psi_j'\,\mathrm{d}x\,.$$

umerical integration utilizing a value of $\sum_k c_k\psi_k$ from a previous iteration
ust in general be used to compute the integral. Now our aim is to integrate
mbolically, as much as we can, to obtain some insight into how the finite element
ethod approximates this term. To be able to derive symbolic expressions, we
ust either turn to the group finite element method or numerical integration in
ie node points. Finite element basis functions φ_i are now used.

roup finite element method. We set $\alpha(u) \approx \sum_k \alpha(u_k)\varphi_k$, and then we
rite

$$\int_0^L \alpha(\sum_k c_k\varphi_k)\varphi_i'\varphi_j'\,\mathrm{d}x \approx \sum_k (\underbrace{\int_0^L \varphi_k\varphi_i'\varphi_j'\,\mathrm{d}x}_{L_{i,j,k}})\alpha(u_k) = \sum_k L_{i,j,k}\alpha(u_k)\,.$$

urther calculations are now easiest to carry out in the reference cell. With P1
lements we can compute $L_{i,j,k}$ for the two k values that are relevant on the
eference cell. Turning to local indices, one gets

$$L^{(e)}_{r,s,t} = \frac{1}{2h}\begin{pmatrix} 1 & -1 \\ -1 & 1\end{pmatrix}, \quad t = 0, 1,$$

here $r, s, t = 0, 1$ are indices over local degrees of freedom in the reference cell
$= q(e, r)$, $j = q(e, s)$, and $k = q(e, t)$). The sum $\sum_k L_{i,j,k}\alpha(u_k)$ at the cell
vel becomes $\sum_{t=0}^1 L^{(e)}_{r,s,t}\alpha(\tilde{u}_t)$, where $\tilde{u}_t$ is $u(x_{q(e,t)})$, i.e., the value of u at local
ode number t in cell number e. The element matrix becomes

$$\frac{1}{2}(\alpha(\tilde{u}_0) + \alpha(\tilde{u}^{(1)}))\frac{1}{h}\begin{pmatrix} 1 & -1 \\ -1 & 1\end{pmatrix}. \tag{53}$$

s usual, we employ a left-to-right numbering of cells and nodes. Row number i
the global matrix gets contributions from the first row of the element matrix

in cell $i-1$ and the last row of the element matrix in cell i. In cell
$i-1$ the sum $\alpha(\tilde{u}_0) + \alpha(\tilde{u}^{(1)})$ corresponds to $\alpha(u_{i-1}) + \alpha(u_i)$. The sa
becomes $\alpha(u_i) + \alpha(u_{i+1})$ in cell number i. We can with this insight a
the contributions to row number i in the global matrix:

$$\frac{1}{2h}(-(\alpha(u_{i-1}) + \alpha(u_i)), \quad \alpha(u_{i-1}) + 2\alpha(u_i) + \alpha(u_{i+1}), \quad \alpha(u_i) + \alpha(u$$

Multiplying by the vector of unknowns u_i results in a formula that
arranged to

$$-\frac{1}{h}(\frac{1}{2}(\alpha(u_i) + \alpha(u_{i+1}))(u_{i+1} - u_i) - \frac{1}{2}(\alpha(u_{i-1}) + \alpha(u_i))(u_i - u_{i-1})$$

which is nothing but the standard finite difference discretization of $-(\cdot$
with an arithmetic mean of $\alpha(u)$ (and the usual factor h because of the int
in the finite element method).

Numerical integration at the nodes. Instead of using the grou
element method and exact integration we can turn to the Trapezoidal
computing $\int_0^L \alpha(\sum_k u_k\varphi_k)\varphi_i'\varphi_j'\,\mathrm{d}x$, again at the cell level since that
convenient when we deal with discontinuous functions φ_i':

$$\begin{aligned}\int_{-1}^1 \alpha(\sum_t \tilde{u}_t\tilde{\varphi}_t)\tilde{\varphi}_r'\tilde{\varphi}_s'\frac{h}{2}dX &= \int_{-1}^1 \alpha(\sum_{t=0}^1 \tilde{u}_t\tilde{\varphi}_t)\frac{2}{h}\frac{d\tilde{\varphi}_r}{dX}\frac{2}{h}\frac{d\tilde{\varphi}_s}{dX}\frac{h}{2}dX \\ &= \frac{1}{2h}(-1)^r(-1)^s\int_{-1}^1 \alpha(\sum_{t=0}^1 u_t\tilde{\varphi}_t(X))dX \\ &\approx \frac{1}{2h}(-1)^r(-1)^s\alpha(\sum_{t=0}^1 \tilde{\varphi}_t(-1)\tilde{u}_t) + \alpha(\sum_{t=0}^1 \tilde{\varphi}\end{aligned}$$

The element matrix in (56) is identical to the one in (53), showing t
group finite element method and Trapezoidal integration are equivale
a standard finite discretization of a nonlinear Laplace term $(\alpha(u)u')'$ u
arithmetic mean for α: $[D_x\overline{x}D_x u]_i$.

Remark about integration in the physical x coordinate.

We might comment on integration in the physical coordinate systen
The common Trapezoidal rule in Section 4.6 cannot be used to inte
derivatives like φ_i', because the formula is derived under the assum
of a continuous integrand. One must instead use the more basic ve
of the Trapezoidal rule where all the trapezoids are summed up. T

straightforward, but I think it is even more straightforward to apply the Trapezoidal rule on the reference cell and assemble the contributions.

The term $\int auv\,\mathrm{d}x$ in the variational form is linear and gives these terms in ıe algebraic equations:

$$\frac{ah}{6}(u_{i-1} + 4u_i + u_{i+1}) = ah[u - \frac{h^2}{6}D_xD_xu]_i\,.$$

he final term in the variational form is the Neumann condition at the boundry: $Cv(0) = C\varphi_i(0)$. With a left-to-right numbering only $i = 0$ will give a ɔntribution $Cv(0) = C\delta_{i0}$ (since $\varphi_i(0) \neq 0$ only for $i = 0$).

Summary.

For the equation

$$-(\alpha(u)u')' + au = f(u),$$

P1 finite elements results in difference equations where

- the term $-(\alpha(u)u')'$ becomes $-h[D_x\overline{\alpha(u)}^{x}D_xu]_i$ if the group finite element method or Trapezoidal integration is applied,
- $f(u)$ becomes $hf(u_i)$ with Trapezoidal integration or the "mass matrix" representation $h[f(u) - \frac{h}{6}D_xD_xf(u)]_i$ if computed by a group finite element method,
- au leads to the "mass matrix" form $ah[u - \frac{h}{6}D_xD_xu]_i$.

As we now have explicit expressions for the nonlinear difference equations lso in the finite element method, a Picard or Newton method can be defined as ıown for the finite difference method. However, our efforts in deriving symbolic ɔrms of the difference equations in the finite element method was motivated by a esire to see how nonlinear terms in differential equations make the finite element nd difference method different. For practical calculations in computer programs e apply numerical integration, normally the more accurate Gauss-Legendre uadrature rules, to the integrals directly. This allows us to easily *evaluate* the onlinear algebraic equations for a given numerical approximation of u (here enoted u^-). To *solve* the nonlinear algebraic equations we need to apply the icard iteration method or Newton's method to the variational form directly, as ıown next.

4.8 Picard iteration defined from the variational fo

We address again the problem (43) with the corresponding variational fo Our aim is to define a Picard iteration based on this variational form with attempt to compute integrals symbolically as in the previous three sectio idea in Picard iteration is to use a previously computed u value in the n functions $\alpha(u)$ and $f(u)$. Let u^- be the available approximation to u f previous iteration. The linearized variational form for Picard iteration

$$\int_0^L (\alpha(u^-)u'v' + auv)\,\mathrm{d}x = \int_0^L f(u^-)v\,\mathrm{d}x - Cv(0), \quad \forall v \in V\,.$$

This is a linear problem $a(u, v) = L(v)$ with bilinear and linear forms

$$a(u,v) = \int_0^L (\alpha(u^-)u'v' + auv)\,\mathrm{d}x, \quad L(v) = \int_0^L f(u^-)v\,\mathrm{d}x - Cv(0$$

Make sure to distinguish the coefficient a in auv from the differential e from the a in the abstract bilinear form notation $a(\cdot,\cdot)$.

The linear system associated with (57) is computed the standard wa nically, we are back to solving $-(\alpha(x)u')' + au = f(x)$. The unknown u i on the form $u = B(x) + \sum_{j\in\mathcal{I}_s} c_j\psi_j$, with $B(x) = D$ and $\psi_i = \varphi_{\nu(i)}$, $\nu(i)$ and $\mathcal{I}_s = \{0, 1, \ldots, N = N_n - 2\}$.

4.9 Newton's method defined from the variational f

Application of Newton's method to the nonlinear variational form (51) from the problem (43) requires identification of the nonlinear algebraic ec $F_i(c_0, \ldots, c_N) = 0$, $i \in \mathcal{I}_s$, and the Jacobian $J_{i,j} = \partial F_i/\partial c_j$ for $i, j \in \mathcal{I}_s$

The equations $F_i = 0$ follows from the variational form

$$\int_0^L (\alpha(u)u'v' + auv)\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x - Cv(0), \quad \forall v \in V,$$

by choosing $v = \psi_i$, $i \in \mathcal{I}_s$, and setting $u = \sum_{j\in\mathcal{I}_s} c_j\psi_j$, maybe with a b function to incorporate Dirichlet conditions.

With $v = \psi_i$ we get

$$F_i = \int_0^L (\alpha(u)u'\psi_i' + au\psi_i - f(u)\psi_i)\,\mathrm{d}x + C\psi_i(0) = 0, \quad i \in \mathcal{I}_s\,.$$

In the differentiations leading to the Jacobian we will frequently use the

$$\frac{\partial u}{\partial c_j} = \frac{\partial}{\partial c_j}\sum_k c_k\psi_k = \psi_j, \quad \frac{\partial u'}{\partial c_j} = \frac{\partial}{\partial c_j}\sum_k c_k\psi_k' = \psi_j'\,.$$

The derivation of the Jacobian of (58) goes as

$$J_{i,j} = \frac{\partial F_i}{\partial c_j} = \int_0^L \frac{\partial}{\partial c_j}(\alpha(u)u'\psi_i' + au\psi_i - f(u)\psi_i)\,\mathrm{d}x$$
$$= \int_0^L ((\alpha'(u)\frac{\partial u}{\partial c_j}u' + \alpha(u)\frac{\partial u'}{\partial c_j})\psi_i' + a\frac{\partial u}{\partial c_j}\psi_i - f'(u)\frac{\partial u}{\partial c_j}\psi_i)\,\mathrm{d}x$$
$$= \int_0^L ((\alpha'(u)\psi_j u' + \alpha(u)\psi_j'\psi_i' + a\psi_j\psi_i - f'(u)\psi_j\psi_i)\,\mathrm{d}x$$
$$= \int_0^L (\alpha'(u)u'\psi_i'\psi_j + \alpha(u)\psi_i'\psi_j' + (a - f(u))\psi_i\psi_j)\,\mathrm{d}x \tag{59}$$

When calculating the right-hand side vector F_i and the coefficient matrix $_{i,j}$ in the linear system to be solved in each Newton iteration, one must use a reviously computed u, denoted by u^-, for the symbol u in (58) and (59). With his notation we have

$$F_i = \int_0^L \left(\alpha(u^-)u^{-\prime}\psi_i' + (a - f(u^-))\psi_i\right)\,\mathrm{d}x - C\psi_i(0), \quad i \in \mathcal{I}_s, \tag{60}$$
$$J_{i,j} = \int_0^L (\alpha'(u^-)u^{-\prime}\psi_i'\psi_j + \alpha(u^-)\psi_i'\psi_j' + (a - f(u^-))\psi_i\psi_j)\,\mathrm{d}x, \quad i,j \in \mathcal{I}_s\,. \tag{61}$$

hese expressions can be used for any basis $\{\psi_i\}_{i\in\mathcal{I}_s}$. Choosing finite element unctions for ψ_i, one will normally want to compute the integral contribution cell y cell, working in a reference cell. To this end, we restrict the integration to one ell and transform the cell to $[-1,1]$. The most recently computed approximation $^-$ to u becomes $\tilde{u}^- = \sum_t \tilde{u}_t^{-1}\tilde{\varphi}_t(X)$ over the reference element, where $\tilde{u}_t^{-1}$ is he value of u^- at global node (or degree of freedom) $q(e,t)$ corresponding to the ocal node t (or degree of freedom). The formulas (60) and (61) then change to

$$\tilde{F}_r^{(e)} = \int_{-1}^1 \left(\alpha(\tilde{u}^-)\tilde{u}^{-\prime}\tilde{\varphi}_r' + (a - f(\tilde{u}^-))\tilde{\varphi}_r\right)\det J\,\mathrm{d}X - C\tilde{\varphi}_r(0), \tag{62}$$
$$\tilde{J}_{r,s}^{(e)} = \int_{-1}^1 (\alpha'(\tilde{u}^-)\tilde{u}^{-\prime}\tilde{\varphi}_r'\tilde{\varphi}_s + \alpha(\tilde{u}^-)\tilde{\varphi}_r'\tilde{\varphi}_s' + (a - f(\tilde{u}^-))\tilde{\varphi}_r\tilde{\varphi}_s)\det J\,\mathrm{d}X, \tag{63}$$

ith $r,s \in I_d$ runs over the local degrees of freedom.

Many finite element programs require the user to provide F_i and $J_{i,j}$. Some rograms, like FEniCS[2], are capable of automatically deriving $J_{i,j}$ if F_i is pecified.

Dirichlet conditions. Incorporation of the Dirichlet values by assembling ontributions from all degrees of freedom and then modifying the linear system

[2] `http://fenicsproject.org`

can be obviously be applied to Picard iteration as that method involves a s linear system. In the Newton system, however, the unknown is a co δu to the solution. Dirichlet conditions are implemented by inserting the initial guess u^- for the Newton iteration and implementing $\delta u_i =$ known degrees of freedom. The manipulation of the linear system follows the algorithm in the linear problems, the only difference being that the values are zero.

5 Multi-dimensional PDE problems

5.1 Finite element discretization

The derivation of F_i and $J_{i,j}$ in the 1D model problem is easily genera multi-dimensional problems. For example, Backward Euler discretizatio PDE

$$u_t = \nabla\cdot(\alpha(u)\nabla u) + f(u),$$

gives the nonlinear time-discrete PDEs

$$u^n - \Delta t\nabla\cdot(\alpha(u^n)\nabla u^n) + f(u^n) = u^{n-1},$$

or with u^n simply as u and u^{n-1} as $u^{(1)}$,

$$u - \Delta t\nabla\cdot(\alpha(u^n)\nabla u) - \Delta t f(u) = u^{(1)}\,.$$

The variational form, assuming homogeneous Neumann conditions for si becomes

$$\int_\Omega (uv + \Delta t\alpha(u)\nabla u\cdot\nabla v - \Delta t f(u)v - u^{(1)}v)\,\mathrm{d}x\,.$$

The nonlinear algebraic equations follow from setting $v = \psi_i$ and us representation $u = \sum_k c_k\psi_k$, which we just write as

$$F_i = \int_\Omega (u\psi_i + \Delta t\alpha(u)\nabla u\cdot\nabla\psi_i - \Delta t f(u)\psi_i - u^{(1)}\psi_i)\,\mathrm{d}x\,.$$

Picard iteration needs a linearization where we use the most recent approx u^- to u in α and f:

$$F_i \approx \hat{F}_i = \int_\Omega (u\psi_i + \Delta t\alpha(u^-)\nabla u\cdot\nabla\psi_i - \Delta t f(u^-)\psi_i - u^{(1)}\psi_i)\,\mathrm{d}x\,.$$

The equations $\hat{F}_i = 0$ are now linear and we can easily derive a linear sy the unknown coefficients $\{c_i\}_{i\in\mathcal{I}_s}$ by inserting $u = \sum_j c_j\psi_j$.

In Newton's method we need to evaluate F_i with the known value u

$$F_i \approx \hat{F}_i = \int_\Omega (u^- \psi_i + \Delta t \alpha(u^-)\nabla u^- \cdot \nabla \psi_i - \Delta t f(u^-)\psi_i - u^{(1)}\psi_i)\,\mathrm{d}x\,. \quad (68)$$

he Jacobian is obtained by differentiating (66) and using $\partial u/\partial c_j = \psi_j$:

$$J_{i,j} = \frac{\partial F_i}{\partial c_j} = \int_\Omega (\psi_j\psi_i + \Delta t\alpha'(u)\psi_j \nabla u\cdot\nabla\psi_i + \Delta t\alpha(u)\nabla\psi_j\cdot\nabla\psi_i - \Delta t f'(u)\psi_j\psi_i)\,\mathrm{d}x\,. \quad (69)$$

he evaluation of $J_{i,j}$ as the coefficient matrix in the linear system in Newton's ıethod applies the known approximation u^- for u:

$$J_{i,j} = \int_\Omega (\psi_j\psi_i + \Delta t\alpha'(u^-)\psi_j \nabla u^-\cdot\nabla\psi_i + \Delta t\alpha(u^-)\nabla\psi_j\cdot\nabla\psi_i - \Delta t f'(u^-)\psi_j\psi_i)\,\mathrm{d}x\,. \quad (70)$$

:opefully, these example also show how convenient the notation with u and $^-$ is: the unknown to be computed is always u and linearization by inserting nown (previously computed) values is a matter of adding an underscore. One an take great advantage of this quick notation in software [2].

Non-homogeneous Neumann conditions. A natural physical flux condi-on for the PDE (64) takes the form of a non-homogeneous Neumann condition

$$-\alpha(u)\frac{\partial u}{\partial n} = g, \quad \boldsymbol{x}\in\partial\Omega_N, \quad (71)$$

here g is a prescribed function and $\partial\Omega_N$ is a part of the boundary of the domain :. From integrating $\int_\Omega \nabla\cdot(\alpha\nabla u)\,\mathrm{d}x$ by parts, we get a boundary term

$$\int_{\partial\Omega_N} \alpha(u)\frac{\partial u}{\partial u} v\,\mathrm{d}s\,. \quad (72)$$

ıserting the condition (71) into this term results in an integral over prescribed alues: $-\int_{\partial\Omega_N} gv\,\mathrm{d}s$. The nonlinearity in the $\alpha(u)$ coefficient condition (71) ıerefore does not contribute with a nonlinearity in the variational form.

:obin conditions. Heat conduction problems often apply a kind of Newton's ɔoling law, also known as a Robin condition, at the boundary:

$$-\alpha(u)\frac{\partial u}{\partial u} = h_T(u)(u - T_s(t)), \quad \boldsymbol{x}\in\partial\Omega_R, \quad (73)$$

here $h_T(u)$ is a heat transfer coefficient between the body (Ω) and its sur-ɔundings, T_s is the temperature of the surroundings, and $\partial\Omega_R$ is a part of the

boundary where this Robin condition applies. The boundary integral (becomes

$$\int_{\partial\Omega_R} h_T(u)(u - T_s(T))v\,\mathrm{d}s,$$

by replacing $\alpha(u)\partial u/\partial u$ by $h_T(u - T_s)$. Often, $h_T(u)$ can be taken as c and then the boundary term is linear in u, otherwise it is nonlinear and con to the Jacobian in a Newton method. Linearization in a Picard metl typically use a known value in h_T, but keep the u in $u - T_s$ as u $h_T(u^-)(u - T_s(t))$.

5.2 Finite difference discretization

A typical diffusion equation

$$u_t = \nabla\cdot(\alpha(u)\nabla u) + f(u),$$

can be discretized by (e.g.) a Backward Euler scheme, which in 2D written

$$[D_t^- u = D_x\overline{\alpha}^x D_x u + D_y\overline{\alpha}^y D_y u + f(u)]^n_{i,j}\,.$$

We do not dive into details of boundary conditions now. Dirichlet and N conditions are handled as in linear diffusion problems.

Writing the scheme out, putting the unknown values on the left-ha and known values on the right-hand side, and introducing $\Delta x = \Delta y = h$ some writing, one gets

$$\begin{aligned} u^n_{i,j} - \frac{\Delta t}{h^2}(&\frac{1}{2}(\alpha(u^n_{i,j}) + \alpha(u^n_{i+1,j}))(u^n_{i+1,j} - u^n_{i,j}) \\ &- \frac{1}{2}(\alpha(u^n_{i-1,j}) + \alpha(u^n_{i,j}))(u^n_{i,j} - u^n_{i-1,j}) \\ &+ \frac{1}{2}(\alpha(u^n_{i,j}) + \alpha(u^n_{i,j+1}))(u^n_{i,j+1} - u^n_{i,j}) \\ &- \frac{1}{2}(\alpha(u^n_{i,j-1}) + \alpha(u^n_{i,j}))(u^n_{i,j} - u^n_{i-1,j-1})) - \Delta t f(u^n_{i,j}) = u^n_{i,} \end{aligned}$$

This defines a nonlinear algebraic system $A(u)u = b(u)$. A Picard i applies old values u^- in α and f, or equivalently, old values for u in A $b(u)$. The result is a linear system of the same type as those arisi $u_t = \nabla\cdot(\alpha(\boldsymbol{x})\nabla u) + f(\boldsymbol{x},t)$.

Newton's method is as usual more involved. We first define the n algebraic equations to be solved, drop the superscript n, and introduce u^{n-1}:

$$
\begin{aligned}
{}_{i,j} = u^n_{i,j} - \frac{\Delta t}{h^2}(&\\
&\frac{1}{2}(\alpha(u^n_{i,j}) + \alpha(u^n_{i+1,j}))(u^n_{i+1,j} - u^n_{i,j}) - \frac{1}{2}(\alpha(u^n_{i-1,j}) + \alpha(u^n_{i,j}))(u^n_{i,j} - u^n_{i-1,j}) +\\
&\frac{1}{2}(\alpha(u^n_{i,j}) + \alpha(u^n_{i,j+1}))(u^n_{i,j+1} - u^n_{i,j}) - \frac{1}{2}(\alpha(u^n_{i,j-1}) + \alpha(u^n_{i,j}))(u^n_{i,j} - u^n_{i-1,j-1}))\\
&\Delta t f(u^n_{i,j}) - u^{n-1}_{i,j} = 0\,.
\end{aligned}
$$

is convenient to work with two indices i and j in 2D finite difference discretizaons, but it complicates the derivation of the Jacobian, which then gets four ıdices. The left-hand expression of an equation $F_{i,j} = 0$ is to be differentiated ith respect to each of the unknowns $u_{r,s}$ (short for $u^n_{r,s}$), $r \in \mathcal{I}_x$, $s \in \mathcal{I}_y$,

$$
J_{i,j,r,s} = \frac{\partial F_{i,j}}{\partial u_{r,s}}\,.
$$

iven i and j, only a few r and s indices give nonzero contribution since $F_{i,j}$ ontains $u_{i\pm1,j}$, $u_{i,j\pm1}$, and $u_{i,j}$. Therefore, $J_{i,j,r,s}$ is very sparse and we can set p the left-hand side of the Newton system as

$$
\begin{aligned}
{}_{i,j,r,s}\delta u_{r,s} = J_{i,j,i,j}\delta u_{i,j} + J_{i,j,i-1,j}\delta u_{i-1,j} + J_{i,j,i+1,j}\delta u_{i+1,j} + J_{i,j,i,j-1}\delta u_{i,j-1}&\\
+ J_{i,j,i,j+1}\delta u_{i,j+1}&
\end{aligned}
$$

he specific derivatives become

$$
\begin{aligned}
J_{i,j,i-1,j} &= \frac{\partial F_{i,j}}{\partial u_{i-1,j}}\\
&= \frac{\Delta t}{h^2}(\alpha'(u_{i-1,j})(u_{i,j} - u_{i-1,j}) + \alpha(u_{i-1,j})(-1))\\
J_{i,j,i+1,j} &= \frac{\partial F_{i,j}}{\partial u_{i+1,j}}\\
&= \frac{\Delta t}{h^2}(-\alpha'(u_{i+1,j})(u_{i+1,j} - u_{i,j}) - \alpha(u_{i-1,j}))\\
J_{i,j,i,j-1} &= \frac{\partial F_{i,j}}{\partial u_{i,j-1}}\\
&= \frac{\Delta t}{h^2}(\alpha'(u_{i,j-1})(u_{i,j} - u_{i,j-1}) + \alpha(u_{i,j-1})(-1))\\
J_{i,j,i,j+1} &= \frac{\partial F_{i,j}}{\partial u_{i,j+1}}\\
&= \frac{\Delta t}{h^2}(-\alpha'(u_{i,j+1})(u_{i,j+1} - u_{i,j}) - \alpha(u_{i,j-1}))
\end{aligned}
$$

he $J_{i,j,i,j}$ entry has a few more terms. Inserting u^- for u in the J formula and ıen forming $J\delta u = -F$ gives the linear system to be solved in each Newton eration.

5.3 Continuation methods

Picard iteration or Newton's method may diverge when solving PDEs wit nonlinearities. Relaxation with $\omega < 1$ may help, but in highly nonlinear p it can be necessary to introduce a *continuation parameter* Λ in the p $\Lambda = 0$ gives a version of the problem that is easy to solve, while $\Lambda = 1$ is th problem. The idea is then to increase Λ in steps, $\Lambda_0 = 0, \Lambda_1 < \cdots < \Lambda_n =$ use the solution from the problem with Λ_{i-1} as initial guess for the itera the problem corresponding to Λ_i.

The continuation method is easiest to understand through an e Suppose we intend to solve

$$
-\nabla \cdot (||\nabla u||^q \nabla u) = f,
$$

which is an equation modeling the flow of a non-Newtonian fluid th channel or pipe. For $q = 0$ we have the Poisson equation (correspond Newtonian fluid) and the problem is linear. A typical value for pseudc fluids may be $q_n = -0.8$. We can introduce the continuation parameter Λ such that $q = q_n\Lambda$. Let $\{\Lambda_\ell\}_{\ell=0}^n$ be the sequence of Λ values in $[0,$ corresponding q values $\{q_\ell\}_{\ell=0}^n$. We can then solve a sequence of proble

$$
-\nabla \cdot \left(||\nabla u||_\ell^q \nabla u^\ell\right) = f, \quad \ell = 0, \ldots, n,
$$

where the initial guess for iterating on u^ℓ is the previously computed $u^{\ell-1}$. If a particular Λ_ℓ leads to convergence problems, one may try a increase in Λ: $\Lambda_* = \frac{1}{2}(\Lambda_{\ell-1} + \Lambda_\ell)$, and repeat halving the step in convergence is reestablished.

6 Exercises

Problem 1: Determine if equations are nonlinear or

Classify each term in the following equations as linear or nonlinear. Assu a and b are unknown numbers and that u and v are unknown functic other symbols are known quantities.

1. $b^2 = 1$
2. $a + b = 1$, $a - 2b = 0$
3. $mu'' + \beta|u'|u' + cu = F(t)$
4. $u_t = \alpha u_{xx}$
5. $u_{tt} = c^2\nabla^2 u$
6. $u_t = \nabla \cdot (\alpha(u)\nabla u) + f(x, y)$
7. $u_t + f(u)_x = 0$

8. $\boldsymbol{u}_t + \boldsymbol{u}\cdot\nabla\boldsymbol{u} = -\nabla p + r\nabla^2\boldsymbol{u}$, $\nabla\cdot\boldsymbol{u} = 0$ ($\boldsymbol{u}$ is a vector field)
9. $u' = f(u,t)$
10. $\nabla^2 u = \lambda e^u$

Exercise 2: Derive a formula

Derive (9) in Section 1.7. Filename: `relaxed_Newton.pdf`.

Problem 3: Experience the behavior of Newton's method

The program `Newton_demo.py`[3] illustrates graphically each step in Newton's method and is run like

```
Terminal> python Newton_demo.py f dfdx x0 xmin xmax
```

Use this program to investigate potential problems with Newton's method when solving $e^{-0.5x^2}\cos(\pi x) = 0$. Try a starting point $x_0 = 0.8$ and $x_0 = 0.85$ and watch the different behavior. Just run

```
Terminal>  python Newton_demo.py '0.2 + exp(-0.5*x**2)*cos(pi*x)' \
           '-x*exp(-x**2)*cos(pi*x) - pi*exp(-x**2)*sin(pi*x)' \
           0.85 -3 3
```

and repeat with 0.85 replaced by 0.8.

Problem 4: Linearize a nonlinear vibration ODE

Consider a nonlinear vibration problem

$$mu'' + bu'|u'| + s(u) = F(t), \tag{74}$$

where $m > 0$ is a constant, $b \geq 0$ is a constant, $s(u)$ a possibly nonlinear function of u, and $F(t)$ is a prescribed function. Such models arise from Newton's second law of motion in mechanical vibration problems where $s(u)$ is a spring or restoring force, mu'' is mass times acceleration, and $bu'|u'|$ models water or air drag.

Rewrite the equation for u as a system of two first-order ODEs, and discretize this system by a Crank-Nicolson (centered difference) method. With $v = u'$, we get a nonlinear term $v^{n+\frac{1}{2}}|v^{n+\frac{1}{2}}|$. Use both a geometric and an arithmetic average for $v^{n+\frac{1}{2}}$. In the latter case, explain how to apply Newton's method to solve the nonlinear equations at each time level.

[3] `http://tinyurl.com/nm5587k/nonlin/Newton_demo.py`

Exercise 5: Find the sparsity of the Jacobian

Consider a typical nonlinear Laplace term like $\nabla\cdot\alpha(u)\nabla u$ discretized by c finite differences. Explain why the Jacobian corresponding to this term same sparsity pattern as the matrix associated with the correspondin term $\alpha\nabla^2 u$.

Hint. Set up the unknowns that enter the difference stencil and f sparsity of the Jacobian that arise from the stencil.
Filename: `nonlin_sparsity_Jacobian.pdf`.

Exercise 6: Newton's method for linear problems

Suppose we have a linear system $F(u) = Au - b = 0$. Apply Newton's to this system, and show that the method converges in one iteration. F `nonlin_Newton_linear.pdf`.

Exercise 7: Differentiate a highly nonlinear term

The operator $\nabla\cdot(\alpha(u)\nabla u)$ with $\alpha(u) = ||\nabla u||^q$ appears in several p problems, especially flow of Non-Newtonian fluids. In a Newton method to carry out the differentiation $\partial\alpha(u)/\partial c_j$, for $u = \sum_k c_k\psi_k$. Show that

$$\frac{\partial}{\partial u_j}||\nabla u||^q = q||\nabla u||^{q-2}\nabla u\cdot\nabla\psi_j\,.$$

Filename: `nonlin_differentiate.pdf`.

Problem 8: Discretize a 1D problem with a nonlinea ficient

We consider the problem

$$((1+u^2)u')' = 1, \quad x\in(0,1), \quad u(0) = u(1) = 0\,.$$

a) Discretize (75) by a centered finite difference method on a uniform

b) Discretize (75) by a finite element method with P1 of equal leng the Trapezoidal method to compute all integrals. Set up the resulting system.
Filename: `nonlin_1D_coeff_discretize.pdf`.

Problem 9: Linearize a 1D problem with a nonlinea ficient

We have a two-point boundary value problem

$$((1+u^2)u')' = 1, \quad x\in(0,1), \quad u(0) = u(1) = 0\,.$$

) Construct a Picard iteration method for (76) without discretizing in space.

) Apply Newton's method to (76) without discretizing in space.

) Discretize (76) by a centered finite difference scheme. Construct a Picard ıethod for the resulting system of nonlinear algebraic equations.

) Discretize (76) by a centered finite difference scheme. Define the system f nonlinear algebraic equations, calculate the Jacobian, and set up Newton's ıethod for solving the system.
ilename: `nonlin_1D_coeff_linearize.pdf`.

'roblem 10: Finite differences for the 1D Bratu problem

/e address the so-called Bratu problem

$$u'' + \lambda e^u = 0, \quad x \in (0,1), \quad u(0) = u(1) = 0, \tag{77}$$

here λ is a given parameter and u is a function of x. This is a widely used model roblem for studying numerical methods for nonlinear differential equations. he problem (77) has an exact solution

$$u(x) = -2\ln\left(\frac{\cosh((x-\frac{1}{2})\theta/2)}{\cosh(\theta/4)}\right),$$

here θ solves

$$\theta = \sqrt{2\lambda}\cosh(\theta/4)\,.$$

here are two solutions of (77) for $0 < \lambda < \lambda_c$ and no solution for $\lambda > \lambda_c$. For $= \lambda_c$ there is one unique solution. The critical value λ_c solves

$$1 = \sqrt{2\lambda_c}\frac{1}{4}\sinh(\theta(\lambda_c)/4)\,.$$

numerical value is $\lambda_c = 3.513830719$.

) Discretize (77) by a centered finite difference method.

) Set up the nonlinear equations $F_i(u_0, u_1, \ldots, u_{N_x}) = 0$ from a). Calculate ıe associated Jacobian.
ilename: `nonlin_1D_Bratu_fd.pdf`.

'roblem 11: Integrate functions of finite element expansions

/e shall investigate integrals on the form

$$\int_0^L f(\sum_k u_k\varphi_k(x))\varphi_i(x)\,\mathrm{d}x, \tag{78}$$

where $\varphi_i(x)$ are P1 finite element basis functions and u_k are unknown coe more precisely the values of the unknown function u at nodes x_k. We intı node numbering that goes from left to right and also that all cells have t length h. Given i, the integral only gets contributions from $[x_{i-1}, x_{i+1}]$. interval $\varphi_k(x) = 0$ for $k < i-1$ and $k > i+1$, so only three basis functi contribute:

$$\sum_k u_k\varphi_k(x) = u_{i-1}\varphi_{i-1}(x) + u_i\varphi_i(x) + u_{i+1}\varphi_{i+1}(x)\,.$$

The integral (78) now takes the simplified form

$$\int_{x_{i-1}}^{x_{i+1}} f(u_{i-1}\varphi_{i-1}(x) + u_i\varphi_i(x) + u_{i+1}\varphi_{i+1}(x))\varphi_i(x)\,\mathrm{d}x\,.$$

Split this integral in two integrals over cell L (left), $[x_{i-1}, x_i]$, and cell R $[x_i, x_{i+1}]$. Over cell L, u simplifies to $u_{i-1}\varphi_{i-1} + u_i\varphi_i$ (since $\varphi_{i+1} = 0$ cell), and over cell R, u simplifies to $u_i\varphi_i + u_{i+1}\varphi_{i+1}$. Make a `sympy` p that can compute the integral and write it out as a difference equatio the $f(u)$ formula on the command line. Try out $f(u) = u^2, \sin u, \exp u$.

Hint. Introduce symbols `u_i`, `u_im1`, and `u_ip1` for u_i, u_{i-1}, and u spectively, and similar symbols for x_i, x_{i-1}, and x_{i+1}. Find formulas basis functions on each of the two cells, make expressions for u on the t integrate over each cell, expand the answer and simplify. You can mak code and render it via `http://latex.codecogs.com`. Here are some app Python statements for the latter purpose:

```
from sympy import *
...
# expr_i holdes the integral as a sympy expression
latex_code = latex(expr_i, mode='plain')
# Replace u_im1 sympy symbol name by latex symbol u_{i-1}
latex_code = latex_code.replace('im1', '{i-1}')
# Replace u_ip1 sympy symbol name by latex symbol u_{i+1}
latex_code = latex_code.replace('ip1', '{i+1}')
# Escape (quote) latex_code so it can be sent as HTML text
import cgi
html_code = cgi.escape(latex_code)
# Make a file with HTML code for displaying the LaTeX formula
f = open('tmp.html', 'w')
# Include an image that can be clicked on to yield a new
# page with an interactive editor and display area where the
# formula can be further edited
text = """
<a href="http://www.codecogs.com/eqnedit.php?latex=%(html_code)s"
 target="_blank">
<img src="http://latex.codecogs.com/gif.latex?%(html_code)s"
 title="%(latex_code)s"/>
</a>
 """ % vars()
f.write(text)
f.close()
```

he formula is displayed by loading `tmp.html` into a web browser.
ilename: `fu_fem_int.py`.

roblem 12: Finite elements for the 1D Bratu problem

/e address the same 1D Bratu problem as described in Problem 10.

) Discretize (12) by a finite element method using a uniform mesh with P1 ements. Use a group finite element method for the e^u term.

) Set up the nonlinear equations $F_i(u_0, u_1, \ldots, u_{N_x}) = 0$ from a). Calculate ıe associated Jacobian.
ilename: `nonlin_1D_Bratu_fe.pdf`.

roblem 13: Derive the Newton system from a variational orm

/e study the multi-dimensional heat conduction PDE

$$\varrho c(T)T_t = \nabla \cdot (k(T)\nabla T)$$

ı a spatial domain Ω, with a nonlinear Robin boundary condition

$$-k(T)\frac{\partial T}{\partial n} = h(T)(T - T_s(t)),$$

t the boundary $\partial\Omega$. The primary unknown is the temperature T, ϱ is the density f the solid material, $c(T)$ is the heat capacity, $k(T)$ is the heat conduction, $h(T)$ a heat transfer coefficient, and $T_s(T)$ is a possibly time-dependent temperature f the surroundings.

) Use a Backward Euler or Crank-Nicolson time discretization and derive the ariational form for the spatial problem to be solved at each time level.

) Define a Picard iteration method from the variational form at a time level.

) Derive expressions for the matrix and the right-hand side of the equation ystem that arises from applying Newton's method to the variational form at a me level.

) Apply the Backward Euler or Crank-Nicolson scheme in time first. Derive Newton method at the PDE level. Make a variational form of the resulting DE at a time level.
ilename: `nonlin_heat_Newton.pdf`.

Problem 14: Derive algebraic equations for nonline heat conduction

Consider a 1D heat conduction PDE

$$\varrho c(T)T_t = (k(T)T_x)_x,$$

where ϱ is the density of the solid material, $c(T)$ is the heat capacity, temperature, and $k(T)$ is the heat conduction coefficient.

Use a uniform finite element mesh, P1 elements, and the group finite method to derive the algebraic equations arising from the heat conducti

a) Discretize the PDE by a finite difference method. Use either a Ba Euler or Crank-Nicolson scheme in time.

b) Derive the matrix and right-hand side of a Newton method applie discretized PDE.
Filename: `nonlin_1D_heat_PDE.pdf`.

Problem 15: Investigate a 1D problem with a continu method

Flow of a pseudo-plastic power-law fluid between two flat plates can be ı by

$$\frac{d}{dx}\left(\mu_0\left|\frac{du}{dx}\right|^{n-1}\frac{du}{dx}\right) = -\beta, \quad u'(0) = 0,\ u(H) = 0,$$

where $\beta > 0$ and $\mu_0 > 0$ are constants. A target value of n may be $n =$

a) Formulate a Picard iteration method directly for the differential e problem.

b) Perform a finite difference discretization of the problem in each iteration. Implement a solver that can compute u on a mesh. Verify t solver gives an exact solution for $n = 1$ on a uniform mesh regardless of size.

c) Given a sequence of decreasing n values, solve the problem for using the solution for the previous n as initial guess for the Picard it This is called a continuation method. Experiment with $n = (1, 0.6, 0$ $n = (1, 0.9, 0.8, \ldots, 0.2)$ and make a table of the number of Picard it versus n.

d) Derive a Newton method at the differential equation level and discre resulting linear equations in each Newton iteration with the finite di method.

e) Investigate if Newton's method has better convergence properties thaı iteration, both in combination with a continuation method.

References

.] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, 1995.

2] M. Mortensen, H. P. Langtangen, and G. N. Wells. A FEniCS-based programming framework for modeling turbulent flow by the Reynolds-averaged Navier-Stokes equations. *Advances in Water Resources*, 34(9), 2011.

Index